**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L — A2

(11) International Publication Number: **WO 98/36518**

(43) International Publication Date: 20 August 1998 (20.08.98)

(21) International Application Number: PCT/IL98/00043

(22) International Filing Date: 29 January 1998 (29.01.98)

(30) Priority Data:
08/790,974 30 January 1997 (30.01.97) US

(71)(72) Applicants and Inventors: AZARYA, Arnon [IL/IL]; Yehuda Street 4, 47311 Ramat Hasharon (IL). AZARYA, Yitzhak [IL/IL]; Mivtza Sinai Street 5, Kiriat Ata 28000 (IL).

(74) Agent: EITAN, PEARL, LATZER & COHEN-ZEDEK; Lumir House, Maskit Street 22, 46733 Herzelia (IL).

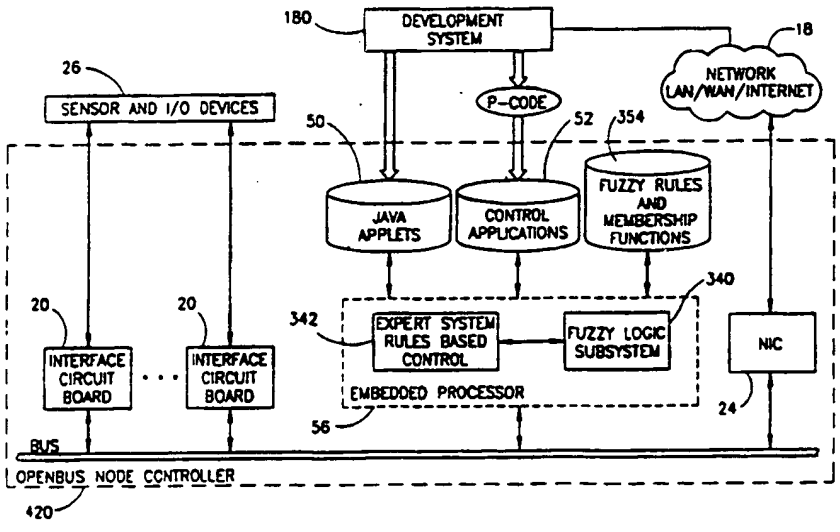
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

**Published**
*Without international search report and to be republished upon receipt of that report.*

(54) Title: OPENBUS SYSTEM FOR CONTROL AUTOMATION NETWORKS INCORPORATING FUZZY LOGIC CONTROL

(57) Abstract

A novel control automation system for enabling I/O boards to access communication networks for receiving and transmitting real time control information over a communication network is disclosed. The system includes a control bus, a node controller and a development system. The node controller includes an expert rule based control system and a fuzzy logic subsystem. The fuzzy logic subsystem is used to perform fuzzy evaluations of control data and variables. The fuzzy control system is defined by the user using fuzzy rules, variables and associated membership functions. A plurality of node controllers connected via a network can send and receive data via agents thus enabling a distributed control application. External hardware that connects to I/O devices such as sensors, motors, monitors, machines, etc. can be connected to the node controller via I/O boards that receive and transmit digital signals, representing control information, to the bus. The bus allows any I/O control board having a common interface, such as ISA, PCI, Compact PCI, etc., to connect to the bus by attachment to one of its slots. An intelligent embedded implementation process provides the logic necessary to enable the connectivity between the I/O boards and the communication network. The development system includes a real-time compiler for generating p-code to be executed on the real-time kernel running in the node controller. The real-time compiler generates p-code from the combination of event triggers, event actions, program logic, fuzzy rules, fuzzy variables and membership functions making up the control application.



*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AL | Albania | ES | Spain | LS | Lesotho | SI | Slovenia |
| AM | Armenia | FI | Finland | LT | Lithuania | SK | Slovakia |
| AT | Austria | FR | France | LU | Luxembourg | SN | Senegal |
| AU | Australia | GA | Gabon | LV | Latvia | SZ | Swaziland |
| AZ | Azerbaijan | GB | United Kingdom | MC | Monaco | TD | Chad |
| BA | Bosnia and Herzegovina | GE | Georgia | MD | Republic of Moldova | TG | Togo |
| BB | Barbados | GH | Ghana | MG | Madagascar | TJ | Tajikistan |
| BE | Belgium | GN | Guinea | MK | The former Yugoslav Republic of Macedonia | TM | Turkmenistan |
| BF | Burkina Faso | GR | Greece | | | TR | Turkey |
| BG | Bulgaria | HU | Hungary | ML | Mali | TT | Trinidad and Tobago |
| BJ | Benin | IE | Ireland | MN | Mongolia | UA | Ukraine |
| BR | Brazil | IL | Israel | MR | Mauritania | UG | Uganda |
| BY | Belarus | IS | Iceland | MW | Malawi | US | United States of America |
| CA | Canada | IT | Italy | MX | Mexico | UZ | Uzbekistan |
| CF | Central African Republic | JP | Japan | NE | Niger | VN | Viet Nam |
| CG | Congo | KE | Kenya | NL | Netherlands | YU | Yugoslavia |
| CH | Switzerland | KG | Kyrgyzstan | NO | Norway | ZW | Zimbabwe |
| CI | Côte d'Ivoire | KP | Democratic People's Republic of Korea | NZ | New Zealand | | |
| CM | Cameroon | | | PL | Poland | | |
| CN | China | KR | Republic of Korea | PT | Portugal | | |
| CU | Cuba | KZ | Kazakstan | RO | Romania | | |
| CZ | Czech Republic | LC | Saint Lucia | RU | Russian Federation | | |
| DE | Germany | LI | Liechtenstein | SD | Sudan | | |
| DK | Denmark | LK | Sri Lanka | SE | Sweden | | |
| EE | Estonia | LR | Liberia | SG | Singapore | | |

# OPENBUS SYSTEM FOR CONTROL AUTOMATION NETWORKS INCORPORATING FUZZY LOGIC CONTROL

## FIELD OF THE INVENTION

The present invention relates generally to computer communication networks and more particularly relates to a system for implementing a control automation network utilizing rule based control in combination with fuzzy logic based decision making.

## BACKGROUND OF THE INVENTION

Openness in the world of automation means being able to buy a variety of products from a variety of vendors and have everything work together seamlessly. To be truly open, however, means the network or platform is accessible to anyone and there is more than one source of enabling technology, i.e., microprocessors and application code. Openness promises significant savings in both time and money. However, recent attempts at openness have not lived up to the promise.

The current market trend is to move to an open, modular architecture controller that will include a horizontal integration of the currently existing fragmented technologies. Currently, most computerized numerical control (CNC), motion and discrete control applications incorporate proprietary control technologies. There are numerous difficulties associated with using proprietary technologies. These include such things as vendor dictated pricing structures, non common interfaces, higher integration costs and the requirement of specific training for troubleshooting and operation. Separate controller elements, a modularity concept and higher level requirements for various elements of an open modular architecture controller are becoming a necessity in a growing number of industries.

The expected benefits of having open and modular architecture controllers include reduced initial investments, low life cycle costs, maximized machine uptime, minimized machine downtime easy maintenance of machines and controllers, easy integration of commercial and user proprietary technologies, plug and play of various hardware and software components, efficient reconfiguration of controllers to support

new processes, incorporation of new technologies and the integration of low cost, high speed communication in machining lines for transferring large amounts of data.

The technology that can enable the new trends and requirements supplied by the personal computer (PC) standards connectivity and communications, the 'Plug and Play' standard for PC cards is becoming a way of life. Within the control industry, the PC is becoming increasingly recognized as a viable technology that will enable the required flexibility and performance.

In today's large automation market, there is a growing number of PC board manufacturers that produce a variety of boards. These boards are targeted towards automation implementation that use the PC and the control platform. Since automation data networks implements a proprietary technology that are not very open for 'Intranet communication.'

Traditional Automation and Control Layer networks are typically medium sized and function to connect PLCs or PCs to related devices within cells or throughout the plant. These networks send small to medium sized packets of data repetitively and have millisecond response times

A high level block diagram illustrating an example prior art proprietary control network including proprietary programmable logic controllers, sensors and I/O devices is shown in Figure 1. A proprietary network 33 (e.g., Fieldbus) forms the core of the automation control system. Connected to this network are programmable logic controllers (PLCs) 34 which as also proprietary. Connected to the PLCs 34 are the sensors and other I/O devices 32. The proprietary PLCs implement the Automation and Control Layer functionality and the sensors and I/O devices implement the Information and Device Layer.

Traditionally, a single manufacturer was able to provide the necessary connectivity with its own network and PLC products and those of qualified third parties. This is not a trivial task as the lead manufacturer must assist these third parties throughout the development process and even after products start to ship. The lead manufacturer, typically the one making the controllers, assumes network ownership by providing specifications, enablers, e.g., chips and software, and test suites for compliance and interoperability.

Examples of previous attempts at openness in the field of industrial networking include the Fieldbus and manufacturing automation protocol (MAP). Both buses are open networks that are not currently meeting user expectations. The MAP bus is not in widespread use today and most vendors have dropped development of MAP products. One of the problems is that although the products have been designed in accordance to a standard specification, many versions of a specification are in use at any one time. In addition, many so called open products require unique configuration software which is only available from the manufacturer of the product. Thus, it becomes a difficult task to get products from different vendors, all built to different versions of a specification, to interoperate together correctly.

Fieldbuses are a special form of local area networks dedicated to applications in the field of data acquisition and the control of sensors and actuators in machines or on the factory floor. Fieldbuses typically operate on twisted pair cables and their performance are optimized for the exchange of short point to point status and command messages. Numerous other Fieldbuses are in existence such as Filbus, Bitbus, FIP, CAN and Profibus standard networks.

Traditional automation and control applications are based around classical binary logic which asserts that every proposition must be either true or false. Recently, however, the use of fuzzy logic has been steadily increasing. Fuzzy logic asserts that each proposition can be true or false as in classical logic but, in addition, asserts that each can also be somewhere in the middle. The use of fuzzy logic within various experts systems is increasing as a way to better model the ambiguousness or vagueness of the real world.

The crisp categories of traditional logic, i.e., 0 or 1, make it difficult when attempting to represent ambiguous events in the real world. Whether designing hardware or software, crisp logic, i.e., classical 0 or 1 logic, forces the user to quantize the world into a discrete set of categories that often do not quite fit the fuzziness of their application. Fuzzy logic overcomes this problem by allowing any shade of gray between 0 and 1 to be directly represented. Each fuzzy parameter has a range of values it can take on, such as slow to fast. In addition, each input has a unique mapping for its range called a membership function.

As mentioned above, traditional logic permits only two truth values, 0 and 1, true or false, yes or no. These crisp values provide only two levels of set membership: either an element is a member of the set or it is not a member. This is a the principle of traditional logic: the law of the excludable middle. Although the law of the excludable middle appears incontrovertible, it is routinely violated by real events in the real world. For example, a valve can often be almost off or the fluid passing through it slightly warm. Real world events can rarely be classified in distinct sets unambiguously.

Traditional logic precisely matches the problem, for example, in pure data processing applications where accuracy is desired when dealing with dollars and cents. However, when dealing with objects in the world, for controlling or modeling real-time events, when interfacing with real people and in numerous other situations, traditional logic is not a good fit. The real world is not packaged into crisp binary packets.

It would be desirable to combine fuzzy logic capability into a control automation system to better approximate the real world. Degrees of set membership, a concept inherent to fuzzy logic, is a conceptualization that allows natural objects in the real world to be directly represented as they exist, while being able to process data with general rules of logic. It permits ranges of membership, such as slow to fast, small to large and light to heavy, to be directly represented, rather than artificially quantizing them merely to be able to process their data.

Previous attempts at incorporating artificial intelligence (AI) have failed in large part due to being forced to use tools that are too precise for the task. Typical AI expert systems use rule sets that grow very large as the application becomes more and more complex. Typical AI systems require hundreds to thousands of precise, crisp if-then rules to mirror the natural fuzziness of the real time world. Equivalent fuzzy logic system, however, typically have tens of rules, an order of magnitude less. This is because the fuzzy parameters in its rules vary so as to apply to a wide variety of cases. In addition, engineers are often requested to solve real world problems with tools of traditional logic that are too tedious and unwieldy. For many engineering problems, high precision is not needed, particularly when machines are to perform tasks previously performed by humans.

Typical applications that are targeted by inference engines, such as real time control, often still require human to observe and react based on experience, e.g., subway train operators. A fuzzy logic inference engine can assist in cases such as these by utilizing a fuzzy rule based expert system.

Fuzzy logic generalizes the Boolean rule 'if X implies Y and X is true, then Y is true.' By substituting infinitely variable parameters for true and false, fuzzy propositions enable different premises to arrive at different conclusions using the same implication rules. Instead of all or none comparisons, a fuzzy processor performs minimum and maximum comparisons. Rather than compare digital 1s and 0s, fuzzy controllers compare variable truth values using minimum and maximum comparison operations.

Fuzzy logic permits an engineer to design expert system like applications which rival the performance of far more complicated AI applications built by specially trained knowledge engineers. Fuzzy logic utilizes ordinary language concepts which are inherently fuzzy rather than any specialized vocabulary. Designing an application using fuzzy logic should be easier than with traditional techniques since the ordinary language concepts that described a problems solution do not have to translated into crisp digital programs to perform meaningful control functions. Further, fuzzy logic control systems are typically smoother due to their inherent intelligence.

Using fuzzy logic, a designer can use fuzzy concepts which apply to an entire range of performance and scale the system's actions to smooth out its operation. The designer does not need to define each situation explicitly, rather the whole range of the system's performance is represented by the membership function, that shows the appropriate response in every case. The fuzzy rules are derived from ordinary language under control of a membership function. For example, a rule in an AI expert system might be "If the pressure is X, then Y the feed valve." A separate rule would be needed for each particular X value considered in the system. In an equivalent fuzzy logic based system, the same rule can be used with X and Y replaced by appropriate membership functions. Thus, the same rule can replace tens or hundreds of crisp quantized rules in the AI system. Utilizing fuzzy rules such as these, a fuzzy knowledge base can be created for an application without requiring a knowledge engineer.

## SUMMARY OF THE INVENTION

The present invention comprises a novel control automation system for enabling I/O boards to access communication networks for receiving and transmitting real time control information over a communication network. The system combines a rule based expert system with a fuzzy logic subsystem which implements a fuzzy logic inference engine. The system includes a control bus, a node controller and a development system. External hardware that connects to I/O devices such as sensors, motors, monitors, machines, etc. can be connected to the invention via I/O boards that receives and transmit digital signals, representing control information, to the bus. The bus functions as the hub of operation, receiving network communications, processing cooperative logic and transmitting information over the communication network. The bus enables single or multiple controllers to access real time information generated by the attached hardware. The bus also enables the execution of I/O operations that originated in external controllers and transmitted over the communication network. The bus allows any I/O control board having a common interface, such as ISA, PCI, Compact PCI, etc., to connect to the bus by attachment to one of its slots. An intelligent embedded implementation process provides the logic necessary to enable the connectivity between the I/O boards and the communication network.

The development system includes a real-time compiler for generating p-code to be executed on the target system. The compiler also generates the code required to implement the fuzzy logic subsystem including the fuzzy rules, variables and membership functions as defined by the control application. The target system, e.g., the node controller, runs the real-time kernel. The target system can be a PC running a commercially available operating system such as Windows NT, VxWorks, Lynx, etc. The real-time compiler generates p-code from the combination of event triggers, event actions and program logic making up the user's application. External input signals and entities such as variables, timers, etc. are analyzed and used to trigger events in the real-time kernel. Based on the program logic as expressed in the p-code, various actions are taken in response to changes in the values of the external input signals and/or entities. The real-time kernel functions to implement a state machine that receives inputs

and generates outputs. The actions taken by the system are represented as a sequence of frames with each frame representing a unit of action.

Changes in the value of external input signals and/or entities trigger one or more events. Each event points to an action, i.e., a set of frames. These actions are then analyzed and executed.

The present invention can be adapted to provide a comprehensive environment for development and execution of fuzzy control application. The system combines rule based control with a fuzzy expert subsystem. The deterministic rule based methodology of the present invention can be used to control an application and the fuzzy expert subsystem can be used to provide fuzzy expert capabilities.

The system is comprised of two subsystems (1) an expert rule based control subsystem and (2) a fuzzy logic subsystem. Both can interface to each other to provide enhanced intelligent control. The expert system can assign a value to one or more fuzzy variables. The crisp input is fuzzified before being processed by the fuzzy logic processor. The expert system may include as part of its rules and actions, an evaluation of one or more fuzzy variables. Once the expert system triggers an evaluation of a fuzzy variable, the fuzzy logic processor performs a fuzzy evaluation of the variable. After processing, a defuzzification process generates crisp output which is passed back to the expert system.

For more information on the theory of fuzzy logic control systems, the reader is directed to A Course in Fuzzy Systems and Control, Wang, Li-Xin, Prentice Hall, 1997, chapters 1 through 4, incorporated herein by reference.

There is therefore provided in accordance with the present invention a control automation system for controlling a plurality of input and output (I/O) devices in accordance with a control application, the system connected to a network for communicating control automation information, the system comprising a development system optionally coupled to the network, the development system generating p-code embodying event triggers, event actions, program logic and fuzzy rules, variables and membership functions implementing the control application and at least one node controller coupled to the network for executing in real-time the p-code generated by the development system and for implementing a fuzzy logic inference engine.

The node controller comprises processor means for managing and controlling the operation of the node controller. The processor means for executing a real-time kernel, the kernel implementing the control application embodied in p-code and implementing the fuzzy logic-inference engine. Also included are network interface means for connecting the node controller to the network, I/O device interface means for connecting the node controller to the plurality of I/O devices and bus means for interconnecting together the real-time kernel, the network interface means and the I/O interface means.

The node controller comprises means for implementing an expert rule based control system and means for implementing the fuzzy logic inference engine. Further, the development system comprises a compiler for generating p-code, machine code, code adapted to be executed on math coprocessor means, code realizing one or more fuzzy evaluation triggers and code including precalculated data of the fuzzy universe of discourse, all in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of the control application.

The kernel means comprises an external input signal scanner for reading, storing and determining changes to external input signals received from the plurality of I/O devices, an event triggers evaluation module for detecting changes to the external input signals and internal entities, the event triggers evaluation module for determining and resolving all event triggers corresponding to the detected changes, a scheduler for marking all actions corresponding to the event triggers that resolve true, an action execution unit for executing and implementing the actions marked for execution by the scheduler, an entity processor for determining any changes to values assigned to an entity, the entity processor notifying the event triggers evaluation module of the entity value changes and means for determining when a fuzzy evaluation is required to be performed.

There is also provided in accordance with the present invention a node controller apparatus for use in a control automation system, the system for controlling a plurality of input and output (I/O) devices in accordance with a control application, the system including a network for communicating control automation information, the apparatus comprising processor means for managing and controlling the operation of the node controller, the processor means for executing a real-time kernel, the kernel implementing

the control application embodied in p-code and for implementing a fuzzy logic subsystem, network interface means for connecting the node controller to the network so as to enable the transfer of one or more agents between node controller connected to the network, I/O interface means for connecting the node controller to the plurality of I/O devices and bus means for interconnecting together the processor means, the kernel means, the network interface means and the I/O interface means.

The processor means comprises means for implementing an expert rule based control system and means for implementing the fuzzy logic inference engine. The processor means may also comprise an expert rule based control system adapted to receive a plurality of input sensor signals from one or more input sensors and adapted to output a plurality of output sensor signals to one or more actuators and a fuzzy logic subsystem for implementing a fuzzy logic inference engine, the fuzzy logic subsystem adapted to receive crisp input data from the expert rule based control system, perform fuzzy processing on the input data to yield crisp output data.

In addition, the fuzzy logic subsystem comprises fuzzification means for transforming the crisp input to fuzzy input, fuzzy logic processing means for performing fuzzy calculations and processing of the fuzzy input and generating fuzzy output in response thereto, defuzzification means for transforming the fuzzy output into crisp output and means for operating the fuzzy logic processing means in accordance with the fuzzy rules, fuzzy variables and associated fuzzy membership functions contained within the control application.

There is further provided in accordance with the present invention a control automation system for implementing a distributed control application, the system connected to a communication network, the system comprising a plurality of node controllers, each node controller comprising expert rule based control system means for implementing the control application in accordance with one or more control rules, fuzzy logic subsystem means for implementing a fuzzy logic inference engine, the fuzzy logic inference engine for evaluating fuzzy variables, expressions and formulas in accordance with one or more fuzzy rules, fuzzy variables and associated membership functions, network interface means for connecting the node controller to the network so as to enable the transfer of one or more agents between node controllers connected to the

network, I/O interface means for connecting the node controller to a plurality of I/O devices and bus means for interconnecting together the expert rule based control system means, the network interface means and the I/O interface means.

The fuzzy logic subsystem comprises fuzzification means for transforming the crisp input to fuzzy input, fuzzy logic processing means for performing fuzzy calculations and processing of the fuzzy input and generating fuzzy output in response thereto, defuzzification means for transforming the fuzzy output into crisp output and means for operating the fuzzy logic processing means in accordance with the fuzzy rules, fuzzy variables and associated fuzzy membership functions contained within the control application.

In addition, there is provided in accordance with the present invention, in a control automation system connected to a communication network, a method of implementing a distributed control application, the method comprising the steps of providing a pluraiity of node controllers wherein each node controller includes expert rule based control system means for implementing the control application in accordance with one or more control rules and fuzzy logic subsystem means for implementing a fuzzy logic inference engine, evaluating fuzzy variables, expressions and formulas in accordance with one or more fuzzy rules, fuzzy variables and associated membership functions making up the control application, connecting the node controller to the network so as to enable the transfer of one or more agents between node controllers connected to the network, connecting the node controller to a plurality of I/O devices and interconnecting together the expert rule based control system means, the network interface means and the I/O interface means.

The step of evaluating comprises the steps of detecting, in accordance with the control rules, that a fuzzy evaluation is to be performed, fuzzifying crisp input output by the expert rule based control system means to yield a fuzzy input, performing a fuzzy evaluation of the fuzzy input in accordance with the fuzzy rules, fuzzy variables and associated membership functions making up the control application to yield a fuzzy output, defuzzifying the fuzzy output to yield crisp output and sending the crisp output results to the expert rule based control system.

Further, there is provided in accordance with the present invention, in a computer system, a method of generating p-code for execution on a node controller as part of a control automation system for controlling a plurality of input and output (I/O) devices in accordance with a user's application, the application including event triggers, event actions and program logic, the method comprising the steps of generating a plurality of pointer tables, each pointer table associated with either an external input signal or an entity, each pointer table comprising a plurality of pointer entries, each pointer entry pointing to an event trigger, generating an event trigger table, the event trigger table comprising a plurality of event trigger entries, each event trigger entry corresponding to an action that references the particular external input signal or entity that points thereto, generating a plurality of actions, each of the actions comprising at least one frame, the actions, the actions representing the generation of output signals and/or the modification of the internal entities and wherein the plurality of pointer tables, the event trigger table and the plurality of actions generated in accordance with the event triggers, event actions and program logic making up the user's application.

There is also provided in accordance with the present invention a kernel for implementation on a computing means, the computing means part of a control automation system for controlling a plurality of input and output (I/O) devices in accordance with a control application, the computing means including an expert rule based control system and a fuzzy logic subsystem, the kernel comprising an external input signal scanner for reading, storing and determining changes to external input signals received from the plurality of I/O devices, an event triggers evaluation module for detecting changes to the external input signals and internal entities, the event triggers evaluation module for determining and resolving all event triggers corresponding to the detected changes, a scheduler for marking all actions corresponding to the event triggers that resolve true, an action execution unit for executing and implementing the actions marked for execution by the scheduler, an entity processor for determining any changes to values assigned to an entity, the entity processor notifying the event triggers evaluation module of the entity value changes, means for determining when a fuzzy evaluation is required to be performed and means for sending crisp input to the fuzzy logic subsystem and receiving crisp output in response thereto.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention is herein described, by way of example only, with reference to the accompanying drawings, wherein:

Fig. 1 is a high level block diagram illustrating an example prior art proprietary control network including proprietary programmable logic controllers, sensors and I/O devices;

Fig. 2 is a diagram illustrating the various layers of the OpenBus control automation network of the present invention;

Fig. 3 is a high level block diagram illustrating a control automation network constructed in accordance with an embodiment of the present invention;

Fig. 4 is a block diagram illustrating the open bus node controller of the present invention connected to a network. sensors and I/O devices;

Fig. 5 is a flow diagram illustrating the embedded open bus control process of the present invention;

Fig. 6 is a flow diagram illustrating the embedded system dispatch process of the present invention;

Fig. 7 is a diagram illustrating the bus width versus throughput for some of the buses in common use today;

Fig. 8 is a diagram illustrating the modular portions of the software making up the OpenBus automation system of the present invention;

Fig. 9 is a high level block diagram illustrating the development system environment of the present invention;

Fig. 10 is a block diagram illustrating, in more detail, the development system environment and the target system of the present invention;

Fig. 11 is a block diagram illustrating the real time kernel of the target system in more detail;

Fig. 12 is a flow diagram illustrating the input signal scanner portion of the real time kernel of the target system;

Fig. 13 is a flow diagram illustrating the entity value change processing portion of the real time kernel of the target system;

Fig. 14 is a flow diagram illustrating the event trigger scheduler portion of the real time kernel of the target system;

Fig. 15 is a block diagram illustrating the internal memory representation of the pointer tables used to implement event triggers and internal/external actions;

Fig. 16 is a block diagram illustrating an example of a frame constructed to implement an action;

Fig. 17 is a flow diagram illustrating the execution sequence of a frame;

Fig. 18 is a block diagram illustrating the open bus node controller of the present invention incorporating a fuzzy logic subsystem coupled to the rule based expert system and to a network, sensors and I/O devices;

Fig. 19 is a block diagram illustrating the fuzzy logic subsystem portion of the embedded processor in more detail;

Fig. 20 is a flow diagram illustrating the fuzzy evaluation method performed in the embedded processor;

Fig. 21 is a flow diagram illustrating the methodology to constructing an fuzzy expert system utilizing the OpenBus system of the present invention;

Fig. 22 is a diagram illustrating the various types of output generated by the fuzzy compiler;

Figs. 23A through 23F are diagrams illustrating the contents of the stack during the calculation of an example membership function;

Fig. 24 is a diagram illustrating the linked structure created to update fuzzy output variables when related fuzzy input variables have changed;

Fig. 25 is a flow diagram illustrating when fuzzy output variables are evaluated;

Fig. 26 is a high level block diagram illustrating the use of agents to distribute both expert rule based system processing and fuzzy logic subsystem processing among other OpenBus controller nodes;

Fig. 27 is a flow diagram illustrating the agent handling method of the present invention;

Fig. 28 is a flow diagram illustrating the agent processing method performed when the arrival of an agent has been detected;

Fig. 29 is a block diagram illustrating the passing of agents from one node controller to another; and

Fig. 30 is a block diagram illustrating an example distributed control application utilizing a control server and an I/O client.

## DETAILED DESCRIPTION OF THE INVENTION

The present invention is a system for providing computer operated real-time process control with the means for interacting with an external system. The system combines an expert system ruled based control with a fuzzy logic inference engine. The present invention also provides a development system comprising a computer compiler for generating real-time code executable on a real-time kernel that resides in a target system. In addition, the present invention provides automation control over standard communication networks such as Ethernet and ATM. The system comprises an intelligent network I/O node controller for automation control that has a common interface with external processors and compilers. In addition, the network I/O node controller implements local logic to create an intelligent controller. A key aspect of the present invention is that automation control information can be transmitted on a conventional backbone network using a conventional connectivity protocol without the need for dual networks, i.e., one for standard data and one for automation control information. Further, a control bus in the intelligent network I/O node controller permits the use of off the shelf I/O cards for interfacing the node controller to input and output devices.

As stated previously, since conventional automation networks implement proprietary technologies they are not well suited for open Intranet communications. The OpenBus system of the present invention functions to fill the void to provide the infrastructure or 'infranet' for communications within a control environment. sensor and actuator level data is managed locally within the infranet but can be shared with higher level data networks through Intranets or other networking platforms. Using open APIs, devices within the infranet share process data and device status information with other nodes via the Intranet or the Internet. The OpenBus system of the present invention enables communication from sensors and actuators on the plant floor to the plant manager's desk anywhere in the world via the Internet, for example, resulting in a seamless network from I/O to the Internet.

OpenBus connectivity can be combined with Java applets in industrial applications making it possible for a plant manager, for example, to monitor, change or

control any element of the industrial control system from the sensor all the way to a high level information system. Plant maintenance personnel can access devices at any point in the network, gather data and make modifications. Service technicians can download new software to devices in the field using Java applets received through an Intranet or Internet connection. If technical support is required, a direct line can be established with a customer support representative to diagnose and repair devices remotely.

**Utilization of OpenBus**

Today, with the advent of more economical and more powerful microprocessors, the world has flattened and broadened. The factory floor, mirroring the organization in general, has seen the number of levels decrease and the span of control increase. With reference to Figures 1 and 2, this flatter, broader view of the world necessities fewer layers, namely: Information Layer 64, Automation and Control Layer 62 and a Device Layer 60 of Figure 2 rather than the two layers of Figure 1.

The Information Layer 64 comprises computers and associated software derived from a variety of suppliers on a variety of computing platforms. The Information Layer is the link between the automation and information environments via manufacturing information systems (MIS) and manufacturing execution systems (MES). Users choose the computing platform, software and operating system for their particular application.

Openness is required here because no one vendor offers the entire scope of host computers, software and communication interfaces, such as computer cards, bridges, routers and media. Considering industrial automation, Ethernet, primarily TCP/IP, has become a de facto standard for the Information Layer. Users purchase products from multiple vendors expecting openness. Control vendors therefor support Ethernet in their controllers, supervisory software and drivers.

The Automation and Control Layer 62 comprises DCS controllers, programmable logic controllers (PLCs), I/O chassis, dedicated human interfaces, motor drives and PCs. This layer is the core of the architecture that bridges the Information and Device Layers, enabling communication throughout the enterprise. Responses here must be in the order of milliseconds to be considered real time. For the Automation and

Control Layer the driving force is the need for deterministic data delivery between controllers and I/O devices.

These devices, however, are specific in nature and are typically proprietary. Control manufactures have traditionally licensed their architecture to other vendors but only in a guaranteed and controlled manner. Only if a limited number of partners work closely together can performance and interoperability be maintained.

In the Device Layer 60, low end devices that are traditionally hardwired into I/O cards are now networked. Devices are either discrete, e.g., sensors, starters, drives, I/O blocks, etc., or process oriented, e.g., transmitters, transducers, valves, single loop controllers, etc. The wide range of devices requires openness in device layer systems. At this level, no single vendor can possible fill all potential product and application needs. The present invention enables verifiable adherence to an accepted standard in order to ensure product compliance and interoperability.

Devices are less complex at the Device Layer than they are at the Information Layer but they are more diverse. The size and cost to imbed connections in a device are critical at this layer. For example, consider adding a network connection to a $ 70 photoeye. In addition, no single vendor can offer all the possible devices, e.g., sensors and actuators, a user could need. For a true device network, the actual devices must be interoperable from manufacture to manufacturer. An I/O device can be taken from one network and be replaced with an I/O device from another network while the operation of the system behaves the same. The present invention provides this level of interoperability by using standard communication control networks such as Ethernet or FDDI on the one hand and by permitting third party connectivity via standard PC I/O boards using standard PC buses such as PCI, EISA or VME.

## OpenBus of the Present Invention

Networks in an automation control system require varying degrees of openness by virtue of the devices they connect and the functions they perform. As previously discussed, difference architecture layers and devices dictate different degrees of openness. Networks must therefore offer a level of openness compatible with the architecture layer and the devices they connect.

Users, however, purchase control products at the device level, e.g., sensors, actuators, pushbuttons, etc., from a number of different manufactures. For this reason, most vendors develop products that adhere to emerging device level networking standards.

The OpenBus system of the present invention of connecting PC bus architectures to area network buses using an embedded application brings the high speed and high throughput capabilities of the area network buses, e.g., Ethernet, FDDI, ATM, etc., as well as the openness of PC buses, e.g., PCI, ISA, EISA, VME, etc., to the large number of third party I/O control board manufacturers. In addition, numerous control functions are implemented locally on the OpenBus using a distributed and cooperative architecture.

Control functions include, but are not limited to, high speed counters, axis control, continuous analog output, fixed analog output, etc. These control functions are implemented via software executing on the OpenBus on board processor thus obviating the conventional method of installing special hardware for each desired function.

A high level block diagram illustrating an automation control network constructed in accordance with an embodiment of the present invention is shown in Figure 3. At the core of the system, termed OpenBus, is the OpenBus node controller 10. Each OpenBus node controller is connected to the network 18 which may be, for example, a LAN, WAN, the Internet, Intranet or any other suitable data, control or area network. Personal computers 14 are also connected to the network. The PCs function to execute various application programs constituting the Information Layer. In addition, the p-code control application can be executed on PCs as well. A gateway 42 provides connectivity to external networks such as the Internet 40.

Coupled to each OpenBus node 10 are sensors and I/O devices 16. For example, a factory floor 12 may contain one to many hundreds of OpenBus node controllers. Using the system of the present invention, a plant manager 15 located anywhere in the world can monitor and control sensors and I/O devices on the floor of a factory located around the world.

As stated previously, the OpenBus system also comprises a development system 180 that may optionally be coupled to the network 18. The development system can be

hosted by a conventional personal computer or equivalent device. The development system enables a user to generate pseudo-code or p-code that can be loaded onto and executed by a real-time kernel that resides in the OpenBus node controller 10. In addition, the p-code can be executed on a standard PC running any commercial operating system. A more detailed description of the development system is presented later in this document.

As stated previously, openness in an industrial control automation network is very desirable. More specifically, within the network, openness is very important at the Control Layer. In the past, this layer was designed by committees which is difficult resulting in a lack of sufficient openness. The system of the present invention functions to open the Device Layer, e.g., sensors, starters, I/O blocks, etc., and to allow third parties to supply I/O control devices that would be able to connect to the network via the OpenBus system using PC boards built according to standards such as PCI, ISA, EISA, VME, etc.

Conventional area networks such as Ethernet, ATM, FDDI, etc., under the present invention, comprise independent intelligent network nodes. Each OpenBus intelligent node comprises an imbedded processor that functions to intermediate between the I/O boards and the area network. A set of high level APIs can be written that allow each processor, controller or computer connected to the area-network or the Internet to access sensor information at the application connectivity layer. In addition, the intelligent nodes participate in a distributed processing control environment by implementing independent local functionality that was previously programmed using the development system.

A high level block diagram illustrating the open bus node controller 10 of the present invention connected to a network, sensors and I/O devices is shown in Figure 4. The node controller 10 comprises one or more interface circuitry boards 20 coupled to a bus. These interface circuitry boards can be any widely available off the shelf third party automation control I/O board designed for either generic or specific applications. The bus can be any commonly used generic conventional bus, such as any of the buses discussed below. A network interface card (NIC) 24 provides the interface circuit boards connectivity to the network 18.

An embedded processor 22 controls and manages the node controller, functioning to control the communication between the interface circuitry boards and the NIC. The embedded processor is capable of executing Java applets 50 and application p-code control applications 52 developed on the development system. The local bus permits certain portions of an application program to be implemented in the node controller as a form of distributed or cooperative automated control processing. Further, the NIC and the I/O boards permit the local attachment of various analog and digital sensors, thus creating an integrated smart sensor attached to the network.

The development system 180 is shown in Figure 4 to illustrate that the p-code it generates forms the intelligent software control means for the embedded processor 22. The process control algorithms and logic flow input by the user using the development system is represented in the p-code that is executed in the OpenBus node controller.

The OpenBus system of Figures 3 and 4 enables I/O boards to access conventional communication networks for the receipt and transmission of real-time control information over the network. Hardware that includes I/O attachment (e.g., sensors, motors, monitors, machines, etc.) are connected to the OpenBus via I/O boards that receive/transmit signals representing control information over the bus.

The OpenBus node controller 10 functions as the hub of operation of the system. It receives network communications, processes cooperative logic and transmits information over the network. The system permits multiple controllers, such as PCs 14 or node controllers 10, to access real time information generated by attached hardware 26 located anywhere in the network. A node controller 10 can execute I/O operations that originated in PCs 14 or other controllers 10 and transmitted over the network 18 and/or the Internet 40 communication networks.

The 'open control' approach implemented in the OpenBus architecture of the present invention provide third party vendors with the required variety and the critical mass of products to satisfy user application needs. An OpenBus control network functions to deliver tested and certified multi-vendor performance from competing third parties, which can only benefit users.

The OpenBus of the present invention satisfies the demand today for more open automation systems in both networks and supporting devices. There is a trend towards

open protocols, e.g., Ethernet, ATM, etc., at the Information Layer. The present invention provides a system able to offer accepted and supported networks at the Device Layer as well. The Automation and Control Layer will remain controlled-open due to the unique performance requirements. As shown in Figure 7, the advancement in speed and throughput within the open network architecture is much advanced in comparison with conventional existing proprietary control networks supplied by a relatively few individual vendors. The OpenBus of the present invention enables the delivery of the latest high speed area network capabilities to the automation control field.

A key feature of the OpenBus system is that it is completely open at the Information and the Device Layers and controlled open at the Automation and Control Layer. A key advantage of this feature is that it offers numerous benefits to users. A user can choose the low end devices and host platforms that best meet application requirements. This provides users with an open architecture whereby the devices are the most variable yet maintain stability over the real time control system. The Automation and Control Layer functions to effectively bridge the Information and Device Layers while maintaining time critical communications between controllers and I/O devices.

**OpenBus Control Processes**

A high level logic flow diagram illustrating the embedded open bus control process of the present invention is shown in Figure 5. With reference to Figures 3 and 4, the first action performed by the embedded processor 22 upon power up is to initialize the node controller (step 70). Once initialized, the processor loads the embedded program into memory (step 72). The embedded program comprises control programs developed by the development system written and compiled into p-code. In addition, the development system can generate Java scripts or applets. Then, the processor goes out on the bus and identifies each of the I/O boards 20 installed on the bus (step 74). Once the I/O boards are identified, the processor attempts to establish communications with the attached network 18 via NIC 24 (step 76).

At this point two separate processes are began which execute is parallel with each other. One process manages the communications over the network and the other process

executes the real-time p-code control application that was previously developed using the development system.

The first step of the network communication management process is to wait for a network communication (step 78). Once a network communication is received, the processor checks if it is a network message (step 80). If it is not a network message control returns to step 78. If it is a network message, the message is then analyzed (step 82) and the dispatcher is activated (step 84). Note that optionally, the network communication management process can be implemented in Java code. The dispatcher is described in more detail below.

The first step of the control application is to load the p-code from an external storage device (step 86). Once the p-code is loaded, it is executed in order to enable and perform the control application logic (step 88).

Note that the node controller can operate as a hub only, tying the sensors and I/O devices 26 (Figure 4) to the network 18 via one or more off the shelf interface circuitry boards 20 without the functionality of executing user's application code (i.e., p-code) and/or Java scripts. Likewise, the node controller can operate to only execute user's application code and/or Java scripts without interfacing sensors and I/O devices to the network.

The dispatcher process of the present invention will now be described in more detail. A high level logic flow diagram illustrating the embedded system dispatch process is shown in Figure 6. The first step performed is to analyze the network request contained in the message (step 90). If the network request is a cooperative processing requires, the processing parameters of the request are parsed (step 102). In accordance with the parameters parsed, an embedded intelligent process is than activated (step 104). This embedded intelligent process then performs reads and/or writes to the I/O boards (steps 106, 108).

If the network request is not a cooperative processing request, it is checked whether the request is an input status request (step 94). If it is, the corresponding data is retrieved from the I/O boards and sent to the requester over the network via the NIC (step 96).

If the request is not an input status request, it is checked whether the request is an output I/O request (step 98). If it is not, control returns to step 90. If it is an output I/O request, the I/O data sent over the network is written to the appropriate I/O board(s) (steps 100, 108).

**Conventional Open Buses**

The level of openness required for an application varies with and is dependent upon the functionality of the communication layer and the types of devices found at that layer. Openness is usually achieved by the use of standards. These standards are either sanctioned by an official body, e.g., IEC/ISA SP50 Fieldbus, or is commonly accepted enough to become a de facto standard, e.g., Ethernet TCP/IP. Many vendors and end users prefer de facto standards over official standards because they result in a shorter time to market and have a singular customer and application focus.

Communications in a signal processing system is performed by buses of various types: direct point to point, shared multi-drop or network (i.e., being made up of links, buses pr switches), control buses, data buses, test and maintenance buses, area network buses, etc. These buses can be implemented in serial fashion (i.e., one data line or fiber) or parallel fashion (i.e., multiple data lines or fibers). They can be slow (e.g., kilobits per second) or fast (e.g., gigabits per second). They may also have protocols varying from simple clocking to elaborate access, validation and acknowledgment schemes. The interface point of either is a computational element or a bus. Examples of conventional open system bus standards available today are illustrated in Figure 7 and described in more detail below. The bus width versus throughput for many buses in common use today are presented. The broad downward sloping arrow indicates the preferred path of bus development, i.e., to faster and narrower buses. These buses can be used to perform different functions, such as system control, data transfer, test and maintenance, input/output (I/O) and area networking. The area networking buses, e.g., Ethernet, 100Base Ethernet, FDDI and ATM, are non proprietary in nature and have an order of magnitude higher throughput when compared with the Fieldbus technology which exists today within automation control bus technology.

Some examples of conventional open system bus standards in common use today will now be described in further detail.

## Control Buses

Control buses are typically used to allow multiple processors to interoperate in a system through the exchange of commands and some data. In small systems, where data traffic is minimal, this single type of bus may be the only bus employed. Some currently available control buses are described briefly below.

**Filbus**

The Filbus is based on distributed intelligence and peer to peer communication. Firmware functions are built into each Filbus I/O module and enable basic capabilities such as pulse count, delay before action and sending/receiving messages to/from other modules on the network. The Filbus runs at 375 Kbps, permits a maximum of 250 nodes, uses master/slave arbitration, uses twisted pair cable and has application in data acquisition.

**Bitbus**

The Bitbus was originally introduced by Intel Corporation as a way to add remote I/O capability to Multibus systems. This original Fieldbus is one of the most mature and most broadly used networks today. Bitbus permits programs to be downloaded and executed in a remote node providing for distributed system configuration. The Bitbus runs at 375 Kbps, permits a maximum of 250 nodes, uses master/slave arbitration, uses twisted pair cable and has application in process control.

**Worldfip**

The Worldfip provides a deterministic scheme for communicating process variables. Worldfip uses an original mechanism whereby the bus arbitrator broadcasts a variable identifier to all nodes on the network, triggering the node producing that variable to place its value into the network. This feature eliminates the notion of node address and makes it possible to design distributed process control systems. The Worldfip runs at 1 Mbps, permits a maximum of 250 nodes, uses a bus arbiter for arbitration, uses twisted pair cable and has application in real time control.

**Profibus**

The Profibus is a Fieldbus network designed for deterministic communication between computers and PLCs. It is based on a real time capable asynchronous token bus principle. Profibus defines multi-master and master slave communication relations, with cyclic or a cyclic access, permitting transfer rates of up to 500 Kbps. The physical layer 1 (2-wire RS-485), the data link layer 2 and the application layer are standardized. Profibus distinguishes between confirmed and unconfirmed services, allowing process communication, broadcast and multitasking. The Profibus runs at 500 Kbps, permits a maximum of 127 nodes, uses token passing for bus arbitration, uses twisted pair cable and has application in inter-PLC communication.

**CAN**

The controller area network (CAN) is a serial bus that is designed to provide an efficient, reliable and very economical link between sensors and actuators. CAN uses a twisted pair cable to communicate at speed of up to 1 Mbps with up to 40 devices. It was originally developed to simplify the wiring in automobiles but its use has spread to machines and factory automation products because of its useful features. Some of its features include the ability of any node to access the bus when the bus is quiet, non destructive bit wise arbitration to allow 100% use of bus bandwidth without loss of data, multimaster, peer to peer and multicast reception, automatic error detection, signaling and retries and data packets of 8 bit length. CAN is the basis of several sensor buses such as DeviceNET from Allen Bradley, CAN Application Layer from CAN in Automation or Honeywell's SDS. The CAN runs at 1 Mbps, uses CSMA for bus arbitration, uses twisted pair cable and has application in sensors and actuators.

**Futurebus+**

The Futurebus+ operates at speeds of 3.2 GBps using 256 parallel lines or 100 MBps using 32 parallel lines. It was designed primarily as a cache-coherent shared memory bus and also supports large block transfers and message passing. Its intended application was as a migration path for the VMEbus. Besides the increased throughput, Futurebus+ features centralized or distributed mastership arbitration, compelled or packet transfer mode, priority or fairness resource sharing, cache coherence for shared memory

multiprocessing, module live insertion and a Control and Status Register standard software interface.

**PI-bus**

The Parallel Intermodule (PI) bus uses the same basic structure as VMEbus but is adapted for real time, fault tolerant applications such as military mission critical systems. PI-bus is a synchronous, loosely coupled, message passing bus. A node may be master and slave capable or only slave capable. PI-bus uses the same backplane transceiver logic (BTL) interface as Futurebus+. PI-bus emphasizes fault tolerance and is inherently supportive of module level fault containment since it is a loosely coupled bus. It also contains features such as hardware supported intermodule communication containment boundaries, an error management protocol that supports determination of contaminated memory, the ability for software to control access to its memory and explicit software control of intermodule communication. PI-bus has no centralized control, the protocol uses a distributed vie for gaining bus mastership. The PI-bus is a 50 MBps bus using 32 parallel lines. Designers of PI-bus intended the bus operation to be a send and forget interface making it inappropriate as a real time interface in a tightly coupled architecture.

**VME**

The VersaModule Europa (VME) bus is one of the most successful high end commercial buses in use and has become a de facto standard in high performance industrial automation. The VMEbus is a shared mulitdrop bus with each node on the bus plugging into the rack backplane such that its address and data lines connect onto the Data Transfer Bus in parallel with those of all other nodes. Tri-state logic is used such that only one node at a time actively drives the bus, with all other nodes passively monitoring its activity. The VMEbus operates at speeds of 40 MBps using 32 parallel lines. Its intended application is as a commercial backplane control bus for high performance systems.

**VME64**

The VME64 bus operates at speeds of 80 MBps using 64 parallel lines. Its intended application is as an upgrade for the VME bus.

**Data Buses**

Data buses are typically used to augment a control bus with a higher throughput path for transfer of data between processors. To achieve high speed a data bus is usually implemented as a network of point to point unidirectional links. This avoids the various transmission line problems associated with a shared multidrop bus. Some data buses currently available are described briefly below.

SCI

The Scaleable Coherent Interface (SCI) bus specification define a network in which nodes are interconnected with a set of unidirectional point to point links. SCI provides scaleable network bandwidth because data transfers between nodes may occur concurrently rather than sequentially via a shared bus. SCI operates at speeds of 1 GBps using 16 parallel lines and 250 MBps using a serial line. The basic SCI network is a unidirectional ring where each node receives data from its predecessor node and sends data to its successor node. A mesh network is implemented by equipping each node with two SCI ring interfaces: one in the horizontal direction and one in the vertical direction. A crossbar switch network can be implemented where each node interfaces to the switch via a minimal two node ringlet. SCI uses cache coherent protocols to guarantee consistent data even when data is locally cached and modified by multiple processors. SCI uses a distributed directory based protocol where each line of memory is associated with a list of processor sharing that line. Each memory line maintains a pointer to the processor at the head of the list. Use of the SCI bus is intended with heterogeneous parallel processors.

SCX

SCX is an offshoot of SCI that is being developed for use with heterogeneous high performance parallel processors. The SCX bus operates at speeds of 1 GBps using 32 parallel lines. SCX also requires two counter rotating rings with a bypass switch at each node, similar to FDDI, for fault tolerance whereby neighbor nodes can bypass a failed link, reforming the two rings into a single double length ring.

QuickRing

QuickRing is an offshoot of SCI developed for low cost applications such as PCs, workstations and parallel processors. The QuickRing bus is a SCI like bus that operates

at speeds of 200 MBps using 6 parallel lines. QuickRing uses a voucher/ticket protocol, which is from the SCI, to reserve space in the target node queue before transmitting a packet.

**HIC**

The Heterogeneous InterConnect (HIC) bus defines a low cost, scaleable, serial interconnect for parallel system construction. An HIC link is a bi-directional connection between two nodes, composed of a pair of unidirectional connections. The HIC bus operates at speeds of 10 Mbps to 1 Gbps using copper wire, differential twisted pair, fiberoptic and coax cable. Multiple HIC links per node can be used to build a variety of network architectures, including both hierarchical networks and flat or mesh networks. HIC supports self routed systems using wormhole routing where the packet is read on the fly and the packet is forwarded without being stored in the intervening node.

**RACEway**

The RACEway bus is a proprietary bus uses the VME 'P2' connector to access a crossbar switch to provide high speed concurrent data paths between boards in a VME chassis. It operates at speeds of 1280 Mbps using 32 parallel lines. The basic element of the RACEway is the RACE crossbar chip which has six I/O channels. A single crossbar chip can interconnect six nodes and provide up to three concurrent 1280 Mbps communication paths between node pairs. Topologies that can be created include fat-tree, switch ring and mesh. The RACEway is a preemtable circuit switched network. The RACEway uses a compelled protocol in that the receiving node can enforce flow control through the use of the 8-wire control and clocking signals. Data flow is bi-directional but can only go in one direction or the other at a time.

**Test and Maintenance Buses**

Test and maintenance buses are typically used to provide a minimally intrusive path to every hardware module in the system to isolate and debug failures and to possibly reconfigure data flows and computational elements to avoid failed elements. It is usually implemented as a serial, low speed interconnection. This bus can be included as a single bus for non critical systems or as a double redundant bus for mission critical systems. Proper use of a test and maintenance bus often requires the cooperation of the

data and control buses, necessitating some type of controller element. A brief description of a few test and maintenance buses currently available is presented below.

**Serial Bus/FireWire**

The High Performance Serial Bus (HPSB) is similar in function to TM-bus. The HPSB operates at speeds of 6 MBps over a backplane or 40 MBps using two differential signal pairs. Its intended application is as a general purpose interface that can replace a variety of I/O types such as RS-232, RS-422 and SCSI. FireWire, one implementation of HPSB, can carry both synchronous data and isochronous mulitmedia communications.

**TM bus**

The Test and Maintenance bus is a linear, synchronous, multi-drop communication bus which transfers data between a master node and one or more slave nodes residing on a single backplane. It is used to communicate diagnostic control and status information between nodes. The TM-bus protocol supports up to 251 separate addresses plus the broadcast and muiticast addresses. The TM-bus operates at speeds of 0.8 MBps using a serial line. Its intended application is for use with PI bus in military applications.

**MTM**

The Module Test and Maintenance (MTM) bus is a parallel multi-drop bus containing five signal lines: Clock, Control, master Data, Slave Data and Pause Request. The MTM bus is intended to provide connectivity between modules within a box, e.g., interconnect JTAG modules. The bus operates at speeds of 1.2 MBps using a serial line.

**JTAG**

The JTAG bus is a widely used bus for on-module testing. JTAG is a serial bus containing four signal lines: Test Clock, Test Mode Select, Test Data Input and Test Data Output. JTAG defines a Test access Port (TAP) and boundary scan architecture for digital integrated circuitry. The JTAG bus provides a solution to the problem of testing assembled printed circuit boards containing highly complex digital integrated circuits and high density surface mounting assembly techniques. It also provides a means of accessing and controlling design-for-test features built into the digital integrated circuits themselves. JTAG is used internally in most new large IC designs to confirm that each internal component performs its required function, that the components are interconnected in the correct manner, that the components interact correctly and that the

IC performs its intended function. The JTAG bus operating at speeds of 3 MBps using a serial line.

**Input/Output Buses**

Input/Output buses are typically used to collect raw data from sensors and distribute processed data to embedded computer displays. These buses are optimized to transfer large blocks or continuous streams of data with minimal concern for error checking and flow control. Some Input/Output buses currently available are described briefly below.

**Fibre Channel**

The Fibre Channel (FC) bus is a universal interface for data channels that is optimized for the predictable transfer of large blocks of data such as those used in file transfers, disk and tape storage systems, communications and imaging devices. Fibre Channel provides bi-directional point-to-point connections and support for connected and connectionless operations. Fibre Channel transfers asynchronous information in variable length frames, consisting of a 24 byte header followed by up to a 2048 byte payload of data. Fibre Channel can be implemented in a ring network, but is intended primarily for a switched network. One node may be connected to another node but is typically connected to a fabric node. The fabric node is an entry into a switch that provides transparent connection to other system nodes. Fibre Channel can operate on coax, twisted copper pair and both single and multimode fiber. It operates at speeds of 100 MBps over a serial line.

**SCSI**

The Small Computer System Interface (SCSI) bus is widely used in workstations to connect the processor to various peripheral devices such as a disk controller. SCSI device are daisy chained together and obtain access to the bus via distributed arbitration. Standard SCSI used an 8 bit bus and a 4 MHz clock to achieve a 4 MBps data transfer rate. Fast SCSI increases throughput to 10 MBps and Fast Wide SCSI uses 16 bits to achieve 20 MBps. SCSI-2 achieves 40 MBps using 32 bits.

1553B

The Mil-Std-1553B Digital Time Division Command/Response Multiplex Data Bus has a long-standing history in military avionics applications where independent boxes need to be interconnected. It operates at speeds of 0.1 MBps and uses a single coax with transfer coupling to reduce the chance of damage when connecting separate boxes. It is usually implemented in a dual or triple standby mode to prevent the bus from becoming a single point of failure in a mission critical application. The 1553 bus uses a 1 MHz clock and Manchester biphase encoding to convert each 16 bit work into a 20 bit serial stream.

RS-232

The EIA RS-232 bus is a widely used bus providing for a point-to-point interface between a single driver and a single receiver at speeds up to 20 Kbps over distance up to 50 feet. An improved version, RS-423, increases the speed to 100 Kbps, increases the number of receivers to ten and reduces the voltage swing to +/- 4 volts. Its intended application is to interconnect terminal and modem equipment.

RS 422

The RS-422 is the Electronic Industries Association (EIA) EIA-485 bus which operates at speeds of 1 MBps using a serial line. It is similar to RS-423 but uses differential driver signals to increase the transmission speed to 10 Mbps. RS-485 is similar to RS-422 but uses tristate drivers to allow multiple drivers form a shared multi-drop bus.

**Area Network Buses**

Area network buses are typically used to interconnect processing systems located in separate physical boxes. These buses have been optimized in the past for bursty traffic, but in the future will handle isochronous traffic for mulitmedia application as well. Some Area network buses currently available are described briefly below.

ATM

The Asynchronous Transfer Mode (ATM) bus was originally conceived as the switching technology for the telephone industry to handle multimedia data. ATM is a logical layer protocol based on bandwidth partitioning for the transmission of large amounts of data, e.g., real-time audio, computer data, images and video, on shared media, point-to-point,

switched networks. ATM transfers digital information in consecutive cells (packets) of constant length consisting of a 5 byte header following by a 48 byte payload of data. The header defines a virtual path and a virtual channel as well as other network management functions. The ATM protocols allow a node to establish static or dynamic connections with many other nodes. Although ATM is optimized for virtual connection oriented services, it can be used for connectionless services as well. ATM can be mapped on top of various existing physical layers such as SONET, Fibre Channel or FDDI. ATM operates at speeds of 155 Mbps to 2.5 Gbps over a serial line. Its intended application is for use in telecommunications and as a LAN for workstations.

**FDDI**

The Fiber Distributed Data Interface (FDDI) bus is a standard for a local area network with a transmission rate of 100 Mbps using a fiber optic cable as the transmission medium. FDDI implements a dual counter rotating ring topology and uses a token access method for packet transmission. FDDI is sometimes used as a higher speed backbone to interconnect several lower speed Ethernet networks. FDDI consists of three layers: Physical Layer, Medium Dependent (PMD), Physical Layer Protocol (PHY) and Media Access Control (MAC). The PMD layer specifies the digital baseboard point-to-point communication between two nodes in the FDDI ring. A ring consists of a set of nodes and unidirectional transmission medium segments connected in a serial closed loop configuration. The dual ring option consists of two identical ring configurations, where the secondary ring transmits in the reverse direction of the primary ring. The PHY Layer specifies the remaining aspects of the physical layer protocols. These protocols include decoding incoming data, encoding outgoing data and clock synchronization. The MAC Layer specifies fair and deterministic access to the medium, address recognition and generation and verification of frames. Access to the ring is controlled by passing a token around the ring. If a node wants to transmit, it strips the token from the ring, transmits frames and then reinserts the token.

**Ethernet**

Ethernet has become the de facto LAN standard for PCs, workstations and their peripherals. Ethernet uses Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol that allows nodes connected in parallel to transmit data without

the normal bus mastership arbitration. A node that wants to transmit data first listens to see if any other node is currently transmitting data on the cable, if not, it proceeds to transmit. The node then listens to see if its intended transmission is garbled on the cable and if so aborts transmission and waits a small time before trying again. This protocol is popular because it is cheap and easy to manage. The Ethernet bus is limited, however, to a bandwidth of 10 Mbps using a serial line.

**100Base Ethernet**

The 100Base Ethernet bus is a higher speed version of Ethernet which retains the original Ethernet protocol for backward compatibility with existing Ethernet. 100Base Ethernet operates at speeds of 100 Mbps using a serial line. Its intended application is as a migration path for 10 Mbps Ethernet.

**Myrinet**

The Myrinet bus is a low cost, high speed hierarchical switched network that uses a point-to-point bi-directional link, with a clock rate of 40 MHz and a bandwidth of eight lines to achieve a 40 MBps throughput in each direction. Links can connect to nodes or mulitport switches, allowing the formation of various network topologies. Data is transmitted in variable length packets, with cut through routing and reverse flow control.

## PC Buses

PC buses is a category of buses that is of high commercial significance. PC buses are typically used to interconnect peripherals to a desktop or laptop computer. These buses are optimized for the low cost PC environment and perform several functions such as control, data and input/output. They are important because of their high commercial usage. Some PC buses currently available are described briefly below.

**PCI**

The Peripheral Component Interconnect (PCI) is an Intel proprietary standard bus designed to handle faster peripherals such as high resolution video boards, disk controllers and LAN devices. PCI defines a low latency path between the microprocessor local bus and other system components. It uses a 33 MHz, 32 bit data path to achieve a data throughput speed of 800 Mbps in burst transfer mode. PCI also defines multiple bus master arbitration and configuration space for automatic software

setup. The PCI specification also covers the software architecture needed to guarantee plug-and-play compatibility of modules within a system. The intent is to allow a system to automatically configure itself when a new module is plugged into the PCI backplane connector.

**VL Bus**

The VESA's Local (VL) Bus is a Video Electronics Standards Association (VESA) standard operating at speeds of 100 MBps using 32 parallel lines. Its intended application is as a migration path for the EISA bus.

**V-Bus**

The V-Bus is intended for high performance symmetric multiprocessing systems. V-Bus is a 64 bit bus that operates at up to 66 MHz to provide a sustained peak rate of 4 Gbps. The bus is controlled by central arbiter, with a parking mode to increase throughput on a lightly loaded bus.

**ISA**

The Industry Standard Architecture (ISA) bus enjoys huge sales volume due to its use in the PC since the introduction of the 80286 processor chip. The ISA local bus uses an 8 MHz, 16 bit data path to achieve a data throughput speed of 64 Mbps. In newer PC that are based on fast chips such as the Pentium, the ISA is still used for slow peripherals such as fax modems.

**EISA**

The Extended ISA (EISA) bus operates at speeds of 16 MBps using 32 parallel lines. Its intended application is as a migration path for ISA to interconnect 80386 peripherals.

The OpenBus system of the present invention enables a cost effective solution that provides openness and interoperability. Users can choose the network or the bus based on which offers the greatest connectivity for the devices they value the most.

## Modular Implementation

As stated previously, the present invention is a system for providing computer operated real-time process control with the means for interacting with an external system while also providing a development system comprising a computer compiler for generating real-time code executable on a real-time kernel that resides in the target

system. As stated previously, the target system can be a standard PC. An illustration of the modular portions of the software making up the OpenBus automation system of the present invention is shown in Figure 8. The real-time kernel 154 is the heart of the OpenBus system in that the control algorithms and logic flow that the user desires to implement is executed by the real-time kernel in the target system. At the core of the real-time kernel are p-code frames 150. A p-code frame or simply frame represents a unit of action or operation in the system. For example, frames specify operations to be performed on internal entities such as variables or timers, for example.

Surrounding the p-code frames are event triggers and event actions 152. Complex control operations as specified by the user are broken down by the development system into one or more frames to be executed by the real-time kernel. One or more frames combine to constitute event triggers and event actions.

Surrounding the real-time kernel 154 is the operating system (OS) 155. The real-time kernel comprises the necessary operating system interface to allow it to execute on any desired operating system. The layer surrounding the OS includes various functional modules that perform various roles in the OpenBus system. These functional elements comprise a module for interfacing to sensors 158, I/O devices 160, motion related devices 162, computerized numerical control (CNC) devices 164, devices requiring motor control 166 and discrete I/O 168. In addition, functionality is provided to communicate with one or more networks 170. Further, a database module 172 provides the connectivity to a database that is used by the real-time kernel and application programs. A graphic module 74 provides graphics and drawing related functionality and an operator interface 156 provides the user interface for used by an operator of the system.

**Development System and Target System**

A high level block diagram illustrating the development system environment of the present invention is shown in Figure 9. At the core of the development system 180 is a real-time compiler 184. The real-time compiler functions to take as input an application 182 as input by a user and generate p-code 186 that is executable on the real-time kernel 192 in the target system 190.

The real-time kernel in the target system (embodied in the embedded processor 22 in Figure 4) dynamically changes in response to the structures and parameters defined by the user and represented in her/his application program 182.

With reference to Figure 10 the development system environment of the present invention will now be described in more detail. As stated previously, the development system 180 comprises an application 182 provided by the user and a real-time compiler 184. The application comprises one or more event triggers 200, one or more event actions 202 and logic 204. These various elements combine to define the user's control program application.

The target system 190 comprises the real-time kernel 192 which functions to execute the p-code generated by the compiler. The real-time kernel 192 comprises an external input signal scanner 210. an event triggers evaluation module 212, a scheduler 214, an action execution unit 216 and an entity processor 218. The external input signal scanner receives external input signals from sensors and I/O devices. The action execution unit generates the signals that are output to the external world.

The event triggers 200, event actions 202 and logic 204 provide the logic and the mechanism required to characterize the behavior of any real-time process the user desires to implement. The behavior of any real-time system which interfaces with external signals and real world processes can be defined using a process state change methodology. In this methodology, when a process transitions to a new state, an action comprising a particular logic and operation sequence is performed. In more complex control processes which include multiple simultaneous activities, the action which must be performed is based upon a combination of processes rather than simply one.

The real-time kernel schedules the execution of event actions in accordance with the process state changes as reflected by the change entity value changes. Entities include but are not limited to variables, timers, counters and external input signals. These various entities are part of the program control logic making up the user's application. Any change to the value of an entity or any external signal triggers an immediate evaluation of the event trigger that incorporates that particular entity.

The programming logic functions as the basis for the event actions. The programming logic comprises pure logic, calculations, mathematical formulas,

interfacing with sensors, discrete I/O, motion control, database operation, communication i.e., over networks and operator interface graphics.

All of the external information and programming logic defined by the user and embodied in her/his application comprises various elements such as event triggers, event actions, program variables, timers, counters, program logic, sensor information, motion trajectory planning, motion control, etc. All these elements are broken down, defined and represented via the frame p-code generated by the real-time compiler.

The p-code making up a frame is the smallest building block that enables the real-time compiler to generate code that executes with a response time required of a real-time system. The p-code making up frames comprises a precompiled, one step direct pointer to any piece of information or element which is required in order to perform the logic or operation of the frame. The logic or operation is performed on entities which include variables, timers, I/O port information, I/O values, etc. The precompiled direct pointer to the memory location of the particular entity permits rapid access to the values and references of the entities associated with a frame. These memory pointer references can be performed extremely rapidly with minimal delay thus providing the real-time response needed by the application control program. This is in direct contrast to conventional compiled programming systems that typically involve run time memory address calculations, hash table calculations, heap and stack addressing, etc. in order to resolve memory references thus creating a huge overhead not present in the real-time kernel of the present invention.

**Real-Time Kernel**

The real-time kernel will now be described in more detail. A high level block diagram illustrating the real time kernel of the target system in more detail is shown in Figure 11. The real-time kernel 192 comprises an external input signal scanner 210, event trigger evaluation module 212, scheduler 214, action execution unit 216 and an entity processor 218. The system under control 220 received outputs from the action execution unit and generates the external input signals input to the external input signal scanner. The p-code 186 generated by the real-time compiler is utilized by the event trigger evaluation module and the action execution unit.

The event trigger evaluation unit 212 receives external input signals from the system under control 220 via external input signal scanner 210 and various entities, e.g., variables, counters, timers, motion vectors, motion loop data, etc., from the entity processor 218. The event trigger evaluation module determines the next state of the system based on the current state and the values of all input entities. The event trigger evaluation module, in combination with the rest of the real-time kernel, serve to implement a state machine. The evaluation module functions to test conditions using a set of rules derived from the user's application.

During a change of state one or more action frames are generated. These action frames are passed to the scheduler 214. The scheduler determines the order of execution for each action frame and passes frames ready for execution to the action execution unit 216.

The action execution unit processes each frame and implements the action contained therein. Depending on the type of action, one or more entities may be modified. The entity processor performs the modification on the entities in accordance with the output of the action execution unit. The action execution unit also generates various output signals such as, but not limited to, motion control signals, digital signals and analog signals. These output signals effect various components of the system under control and serve to modify the state of the system.

In addition, the execution unit outputs data for controlling the computer graphics used in the operator interface. The commands and data are output to the GUI 222 that makes up part of the operating system host the real-time kernel operates in, when the target system is a standard PC.

Further, the execution unit also outputs command and data for controlling a database that are directed to the database handler 224. The database is used to store various data about the system such as attributes, states, entity related data, etc.

The input signal scanner method performed by the external input signal scanner of the real-time kernel will now be described in more detail. A high level flow diagram illustrating the input signal scanner portion of the real time kernel is shown in Figure 12. The first step of the method is to read the values of the external input signals from the various sensors, I/O devices that are monitored by the system (step 230). The values

read in are then stored for future reference (step 232). All the input values and then scanned or examined and all values that have changed since the previous read are flagged (step 234). In any input signals have changed value, a message is generated and sent to the event trigger evaluation module identifying the input signal and its new value (step 236). This method is repeated over and over in an endless loop. Thus, as the values of the signals input to the system change, these changes are immediately detected and reported to the event trigger evaluation module.

The entity value change method will now be described in more detail. A high level flow diagram illustrating the entity value change processing portion of the real time kernel of the target system is shown in Figure 13. The step of interpreting and executing the 'assign entity' commands contained in frames is performed by the action execution unit (step 240). The entity processor receives the values to be assigned to the entities and determines whether any new values have been assigned to an entity (step 242). If there are new values to be assigned to an entity, the event trigger evaluation module is notified accordingly with the entity and it new value (step 244).

The event trigger scheduler method performed by the event trigger evaluation module and the scheduler unit of the real-time kernel will now be described in more detail. A high level flow diagram illustrating the event trigger scheduler is shown in Figure 14. The first step of the method is to get the values of all external input signals and internal entities in the system (step 250). It is then determined whether any values have changed since the previous reading (step 252). For any values that have changed, the event triggers associated with the value change is determined (step 254). This process is described in more detail later.

Once the event triggers are determined, each trigger is resolved to either true or false (step 256). If an event trigger resolves true, its associated action, as represented by its frame, is scheduled by the scheduler for execution by the action execution unit (step 258).

The event trigger handling of the real-time kernel will now be described in more detail. A high level block diagram illustrating the internal memory representation of the pointer tables used to implement event triggers and internal/external actions is shown in Figure 15. Illustrated in the left portion of the Figure are examples of various the

internal pointer representations for internal entities such as variables 280 and timers 282 and for external input signals 284. Every entity and external input signal in the system is represented by a pointer in memory. The pointers are grouped into tables according to entity type. For example, the pointers 260, 262 associated with two variables are shown grouped in the variables pointer table 280. The pointers 264, 266 associated with two timers are shown grouped in the timers pointer table 282. Similarly, three pointers 268, 270, 272 are shown grouped in the external input signal table 284. The pointer tables for variables, timers and external input signals are shown for illustration purposes only. The pointers maintained for a system depends on the input signals and various entities that the application uses.

All the entities and external input signals that make up an action frame have their corresponding pointers point to the event trigger pointer for that particular action frame. Every entity and input signal pointer points to a particular event trigger entry in the event trigger table 278. Each entity or input signal may point to multiple event trigger entries and multiple entity and input signal pointers may point to the same event trigger entry. For example, variable pointers 260, 262 both point to event trigger entry 274. Variable pointer 262 points to event trigger entries 274, 276. Timer pointer 264 points to event triggers 276, 286. External input signal pointer 268 points to event trigger entry 290.

It is important to note that all the entity and external input signal pointers used by the system are determined and resolved during compile and load time rather than during run time. In addition, the configuration or input and output devices is known a priori at compile time. Thus, entity or input signal addresses do not need to be computed during execution of the application as is the case in conventional systems. This is performed by conventional system using hash tables, pointer lists, variable lists, etc. The present invention requires only a pointer for each entity or input signal directly to point to all the event trigger entries associated with that entity or input signal. This helps enable the control system automation system of the present invention to operate quick enough to execute the application in real-time.

Each entry in the event trigger table points, in turn, to an action. As stated previously, an action is comprised of one or more frames. For example, the event trigger entry 276 is shown to point to the action 300. In the example illustrated in Figure 15, the

action frame 300 comprises a plurality of frames 302. Each frame can be one of three different types. The three types of frames include: a program logic frame, an operation frame or a condition frame. A program logic frame includes such program logic as jumps, both conditional and unconditional, etc. Operation frames perform an action such as A = B * D, or assign a value to an entity or output signal. For example, a frame may assign a value to a timer, a variable or a digital or analog I/O port. A condition frame tests the given entities to be true or false, e.g., IF A = B + C THEN set an output signal.

A high level block diagram illustrating an illustrative example of a frame implementing an action is shown in Figure 16. The action 310 comprises six frames with the first frame 312 being an operation type frame. Frame 312 assigns the value 10 to the variable entity 'A'. Frame 314 assigns the value of variable 'C' to 'B'. The next frame 316 is a conditional type frame. If the value of variable 'A' is greater than that of 'B' control passes to frame 318 which outputs the value of variable 'E' to an output port. Alternatively, control passes straight to frame 320 which is an operation type frame which assigns the value 2 to the variable 'D'. The last frame 322 is a program logic type frame which performs a return function.

A high level flow diagram illustrating the execution sequence of a frame is shown in Figure 17. The method of processing a frame begins with reading the p-code for the next frame to be executed (step 330). The frame is then analyzed in accordance with the instructions contained within the p-code (step 332). Any values of external input signals or internal entities, e.g., variables, timers, etc., that are needed to properly analyze the frame are then read (step 334). The command within the frame is then performed and the appropriate output signals or entity value changes are then generated in accordance with the command (step 336).

For example, the action taken by a frame may generate a digital or analog output signal, a motion control output, e.g., closed loop motion control, a change to a database entry, a message to be transmitted over the network, data to be displayed on the operator's console, etc.

The following is a list of commands that make up the frames of an action. These commands are executed by the action execution unit 216 (Figure 11) in the real-time kernel.

| Frame Commands | | |
|---|---|---|
| | Command | Description |
| 1 | IF <condition> THEN<br><statements><br>ELSE<br><statements><br>END | conditional type frame command |
| 2 | CASE <entity><br><value> <statements><br><value> <statements><br><value> <statements><br>END | switch statement for choosing different action based on the value of a variable |
| 3 | FOR <var> = <value> TO <value> BY <value><br><statements><br>END | construct for implementing loops |
| 4 | TRACE <entity> | trace command to show the values of entities, actions, formulas or subroutines |
| 5 | ASSIGN <entity> = <calculation formula> | assignment command for setting values to entities or ports |
| 6 | TIMER START/STOP | timer command with start and stop options |
| 7 | COUNTER RESET/START/UP/DOWN/NEXT | counter function with options to start, reset, count up, down, etc. |
| 8 | DELAY <value> | delays by specified amount of time |
| 9 | LABEL <identifier> | assigns a label to a command |
| 10 | JUMP <label> | jump to statement having specified label |
| 11 | DOSUB <sub name> | perform subroutine having sub name |
| 12 | BREAK <type> | stop execution according to break type |
| 13 | RETURN <value> | return from subroutine with specified value |
| 14 | DIGITAL OUTPUT ON/OFF <port> | set specified digital output port to either '0' or a '1' |
| 15 | ANALOG OUTPUT <port><value> | set specified analog output port to specified value |
| 16 | MOTION profile | includes motion function supplied by the motion control board |
| 17 | START MOTION <motion-name> | start a motion profile |
| 18 | STOP MOTION <motion-name> | stop a motion profile |
| 19 | EVENT <value=Stop/Abort/recovery> | return the axis event |
| 20 | MOTION STATUS | return the motion status |
| 21 | SET POSITION <axis> <position> | dynamic axis position |

| | | |
|---|---|---|
| 22 | GET POSITION <axis> <position> | dynamic axis position |
| 23 | GET ERROR <axis> <error> | dynamic axis position error planned as compared to actual |
| 24 | SET VELOCITY <axis> <velocity> | dynamic velocity |
| 25 | GET VELOCITY <axis> <velocity> | dynamic velocity |
| 26 | SET ACCEL <axis> <velocity> | dynamic acceleration |
| 27 | GET ACCEL <axis> <velocity> | dynamic acceleration |
| 28 | SET JERK <axis> <velocity> | dynamic jerk |
| 29 | GET JERK <axis> <velocity> | dynamic jerk |

Using the above listed commands, action frames can be generated that implement the user's application 182 (Figure 10). More particularly, the frames making up the actions implement the combination of event triggers, event actions and the logic of the user's application. The real-time compiler 184 functions to generate the p-code that is utilized by the event triggers evaluation module 212 and the action execution unit 216.

**Expert Rule Based Control System With Fuzzy Logic Subsystem**

In a second embodiment of the present invention, the rule based control expert system described hereinabove can be combined with a fuzzy logic inference engine to enhance the performance and modeling accuracy of the system. A block diagram illustrating the open bus node controller of the present invention incorporating a fuzzy logic subsystem coupled to the rule based expert system and to a network, sensors and I/O devices is shown in Figure 18. The node controller 420 is similar to that of Figure 4 and comprises one or more interface circuitry boards 20 coupled to a bus. The interface circuitry boards can be any widely available off the shelf third party automation control I/O board designed for either generic or specific applications. The bus can be any commonly used generic conventional bus, such as any of the buses discussed below. A network interface card (NIC) 24 provides the interface circuit boards connectivity to the network 18.

An embedded processor 56 controls and manages the node controller, functioning to control the communication between the interface circuitry boards and the NIC. The embedded processor 56 comprises a rules based control expert system 342 and a fuzzy logic subsystem 340. The fuzzy logic subsystem 340 utilizes the fuzzy rules and membership functions database 354 in performing fuzzy calculating and analysis. The

embedded processor is also capable of executing Java applets 50 and application p-code control applications 52 developed on the development system. The local bus permits certain portions of an application program to be implemented in the node controller as a form of distributed or cooperative automated control processing. Further, the NIC and the I/O boards permit the local attachment of various analog and digital sensors, thus creating an integrated smart sensor attached to the network.

The node controller 420 combines the flexibility of two methodologies: expert system rule based control with the intelligence of a fuzzy logic subsystem. Thus, the capabilities of the expert rule based control system are extended to provide fuzzy intelligence. During operation, the expert rule based control system is utilized to provide the control functionality of the application and the fuzzy logic subsystem is utilized to provide fuzzy intelligent capabilities.

The entire combined embedded processor can be implemented using available open system components such as commercially available microprocessors, e.g., Intel Pentium, PowerPC, Motorola 68000, etc. The control system application can be developed and simulated utilizing a commercially available operating system such as Windows NT, UNIX, etc. Once the control application has been developed and debugged, an executable boot program can be generated which, in turn, can be loaded and executed directly on the target microprocessor or embedded system.

Described hereinbelow is a methodology that delivers an intelligent control system application that combines a rule based expert system with fuzzy logic inference capabilities. As shown in Figure 19, the expert system is coupled to the input sensors and output actuators while the fuzzy logic subsystem comprises its own separate fuzzy rules and membership functions. More detailed descriptions of fuzzy rules, linguistics and semantics can be found in A Course in Fuzzy Systems and Control, Wang, Li-Xin, Prentice Hall, 1997, chapters 5 and 6, incorporated herein by reference.

A block diagram illustrating the fuzzy logic subsystem portion of the embedded processor in more detail is shown in Figure 19. The embedded processor 56 comprises an expert rule based control system 342 which receives input signal from sensors and other input device 352 and generates output signals to actuators and other output devices 344. The expert rule based control system 342 interfaces with the fuzzy logic subsystem

340. The fuzzy logic subsystem 340 comprises a fuzzy logic processor 350, fuzzification module 346, defuzzification module 348 and a fuzzy rules and membership function database 354.

The operation of the expert rule based control system 342, including the real-time kernel, has been previously described in connection with Figure 1 through 17 and will not be repeated here. The fuzzy logic subsystem and its interface to the expert rule based control system is described hereinbelow.

During operation of the system, the two subsystems, i.e., expert rule based control subsystem and fuzzy logic subsystem, interface which each other to provide enhanced intelligent control. The expert system 342 may, during its operation, assign a value to one or more fuzzy variables. This is denoted as crisp input which is applied to the fuzzification module 346 in the fuzzy logic subsystem 340. The expert system may include as part of its rules and actions, an evaluation of one or more fuzzy variables. Once the expert system triggers an evaluation of a fuzzy variable, the fuzzy logic processor 350 performs a fuzzy evaluation of the variable. Once performed, a defuzzification process 348 is performed on the results of the evaluation. The defuzzification process generates crisp output which is passed back to the expert system. A more detailed description on how to implement a fuzzy inference engine and on the fuzzification and defuzzification process can be found in A Course in Fuzzy Systems and Control, Wang, Li-Xin, Prentice Hall, 1997, chapters 7 through 15, incorporated herein by reference.

A flow diagram illustrating the fuzzy evaluation method performed in the embedded processor is shown in Figure 20. As mentioned previously, the rule based expert system performs event processing on a continuous basis (step 360). During the course of its operation, the expert system may request the evaluation of a fuzzy variable as they may be freely incorporated into the rules and actions associated with a control application. Thus, it is continuously checked whether the evaluation of a fuzzy variable is required to be performed (step 362). If so, the crisp input that is output by the expert system is fuzzified into one or more fuzzy values by the fuzzification module 346 (Figure 19) (step 364). The fuzzy evaluation is then performed by the fuzzy logic processor 350 (step 366). After the evaluation is complete, the fuzzy results are

defuzzified into crisp output before being transmitted back to the expert system (step 368).

To help illustrate the principles of the present invention, an example fuzzy logic subsystem is presented. The following two assumptions are made:

- X and Y are fuzzy input variables that can have the values Low or High
- VELOCITY is a fuzzy output variable that may have the values Slow or Fast

The fuzzy rules for the system are the following:

1. If (X is Low) AND (Y is Low) Then set VELOCITY to Slow
2. If (X is Low) AND (Y is High) Then set VELOCITY to Slow
3. If (X is High) AND (Y is Low) Then set VELOCITY to Slow
4. If (X is High) AND (Y is High) Then set VELOCITY to Fast

The sole control rule for this example is the following:

1. When (MainTimer is DONE) AND (Evaluation {VELOCITY}>70) Then Turn On Actuator A

Note that control rules may reference numerous types of elements such as digital and analog entities, timers, counters, etc. In the above example control rule, a deterministic timer 'MainTimer' and the fuzzy variable VELOCITY have been combined within the rule. The control rule above would be verified as follows:

1. Check whether timer MainTimer is finished
2. Evaluate the fuzzy variable VELOCITY utilizing inference from the four fuzzy rules shown above.

The result is that only when both conditions are met will the associated action, i.e., turning on sensor A, be performed.

The methodology of the fuzzy logic subsystem will now be described in more detail. A flow diagram illustrating the methodology to constructing an fuzzy expert system utilizing the OpenBus system of the present invention is shown in Figure 21. The first step is to define the fuzzy rules, variables and associated membership functions which make up the fuzzy control system application (step 370). Once defined, the rules, variables and membership functions are compiled using a fuzzy logic compiler to generate real time executable code that can be loaded into memory for execution on a processor (step 372). The executable code can be run, for example, on a standard OS

hosted PC or a boot program can be generated therefrom to permit an embedded processor to run the control application, i.e., a runtime version of the fuzzy logic subsystem. Thus, each control application can be considered a miniature OS, with all the requisite functionality contained therein, including the fuzzy logic subsystem and any necessary operating system components such as networking, file I/O, communications, memory management, process scheduler, etc. Once generated, the executable code is loaded into the program memory associated with a processor (step 374). Finally, the fuzzy logic subsystem is activated and its operation is begun (step 376).

Note that the compiler functions to translate the fuzzy rules, variables and membership functions defined in step 370 into machine code than can be executed on a processor. Given the fuzzy rules, variables and membership functions, i.e., source data, one skilled in the compiler art could construct a compiler to transform the source data into machine code. Standard commercially available compiler generators can also be utilized to construct a suitable compiler.

The compiler processes the fuzzy rules, variables and membership functions, which make up the input to the compiler, so as to generate executable code which effects the fuzzy logic subsystem, i.e., the fuzzy control application. In addition, the executable code generated permits the fuzzy logic subsystem to perform fuzzy evaluations and related fuzzy processing substantially in real time, e.g., milliseconds.

A diagram illustrating the various types of output generated by the fuzzy compiler is shown in Figure 22. The fuzzy compiler 380 functions to generate four types of output: (1) optimized machine code 382 (2) math processing code for membership function calculations 384 (3) fuzzy evaluation trigger code 386 and (4) universe of discourse precalculation data 388.

The optimized machine code 382 is output by the compiler utilizing the same methodologies as the rule based expert system as discussed in connection with Figures 9 through 17. The output comprises the fuzzy rules and control logic as optimized machine code ready to execute on the embedded processor. This method enables a high degree of efficiency for execution of real time applications.

The mathematical processing portion 384 of the output comprises machine code for performing membership function calculations. Input to the compiler, in addition to

control rules and variables, are the membership functions associated with the variables. The compiler is operative to process the fuzzy membership functions to generate appropriate machine code that executes on the embedded processor. Preferably, the compiler generates code able to execute on the math coprocessor if the target embedded processor comprises one. This step improves the performance of the control application since the fuzzy membership function calculations represent a large part of the total fuzzy processing. For each membership function, the compiler generates appropriate machine code which can be executed by the math coprocessor (if the embedded processor has one).

The method for code generation of membership function calculations will now be described in more detail. As described previously, the fuzzy compiler functions to read the formulas or equations making up the membership function and generate machine code which can be executed on the moth coprocessor of the embedded processor. During operation of the control application, when a formula or function must be calculated, control passes to the machine code associated with the formula. The machine code generated by the compiler may correspond to any or all of the following:

- mathematical formulas, e.g., sin, exp, etc.
- numbers
- program variables
- digital and/or analog port values
- motion control parameter values, e.g., encoder position, velocity, etc.

In typical control applications, fuzzy membership functions are calculated frequently. Thus, it is preferable that the fuzzy membership function be evaluated as quickly as possible. It is thus preferable that the machine code generated by the compiler be able to execute on the math coprocessor of the embedded processor in order to reduce computation delays.

The following example illustrates the source code that is generated in response to a mathematical formula. In addition, the execution steps performed to evaluate the formula are also illustrated. The use of the stack structure within the math coprocessor of the embedded processor to evaluate the formula eliminates the need to move data within the memory during execution. The example formula is as follows:

$$A+[B-\sin(C\cdot D)]$$

(1)

where

A represents the value read from an analog port associated with an analog sensor.

B represents the encoder value read from an attached servo motor.

C and D represent working variables within the system.

In response to Equation 1, the compiler generates the following machine code which can be executed directly on the math coprocessor. The code generated is executed each time the formula needs to be calculated.

| | |
|---|---|
| FLD A | push A onto the stack |
| FLD B | push B onto the stack |
| FLD C | push C onto the stack |
| FLD D | push D onto the stack |
| Fmul st(0), st(1) | multiply the two upper frames on the stack |
| Fsin | take the sin function of the top of stack frame |
| Fsub st(0), st(1) | subtract the two upper frames on the stack |
| Fadd | add the two upper frames on the stack |

The step by step execution of the above machine code will now be described. Diagrams illustrating the contents of the stack during the calculation of an example membership function are shown in Figures 23A through 23F. In each Figure, the relevant frames 392 of the stack 390 are shown. The results of the stack after execution of the FLD A instruction is shown in Figure 23A. The value A has been pushed onto the top of the stack 390. The results of the stack after execution of the following three FLD instructions is shown in Figure 23B. The first four stack location contain the values D, C, B, A.

The Fmul instruction performs a floating point multiplication on the top two stack locations. The results of the Fmul instruction are shown in Figure 23C. The next instruction Fsin calculates the sin function of the top of stack value. The results of the Fsin instruction are shown in Figure 23D. The Fsub instruction subtracts the top of stack value from the next to top of stack value. The results of the Fsub instruction are shown in Figure 23E. Finally, the Fadd instruction adds the values in the top two stack locations. The results of the Fadd instruction are shown in Figure 23F.

The above example is presented to illustrate the principle of the present invention. One skilled in the electrical arts would appreciate that the actual instructions will vary in accordance with the particular processor (including any on chip math coprocessor) ~~used~~ to construct the control application.

With reference to Figure 22, the third type of output generated by the compiler comprises code fir implementing the fuzzy evaluation triggers 386. Since the fuzzy evaluation process (including fuzzification and defuzzification processes) consumes large amounts of computational resources, it is preferable that the fuzzy evaluation occurs only when a change occurs since the previous evaluation. The variable triggers serve to provide the information needed to determine whether a new fuzzy evaluation is required. If no changes have been detected, the application utilizes the previous evaluation. This insures that the fuzzy logic subsystem performs the evaluation only on an as needed basis.

The mechanism of determining whether a new fuzzy evaluation is required will now be described in more detail. The system utilizes a link list structure to determine which output variables are effected when an input values changes. A diagram illustrating the linked structure created to update fuzzy output variables when related fuzzy input variables have changed is shown in Figure 24. The system uses this linked list structure to determine whether a defuzzification process is required to be performed. The fuzzy input variable 400 has associated with it all the output variables that would be affected by a change in the particular input variable. In this example, four output variables 402, 404, 406,408 are shown linked by pointers to the input variable in a standard forward linked list fashion.

For each fuzzy input variable, the compiler maintains a link list structure of all the fuzzy output variables that need to be recalculated as a result of a change in the fuzzy input variable. During operation, the dependence of output variables on a particular input variable is already known since the dependencies were determined at compile time.

A flow diagram illustrating when fuzzy output variables are evaluated is shown in Figure 25. The fuzzy calculation of a particular fuzzy output variable can be triggered when (1) a corresponding fuzzy input variable changes or (2) when the fuzzy output variable is part of an expert system rule. A fuzzy output variable is evaluated when a

fuzzy input variable is assigned a new value (step 410). Once the control system detects a change in the value of a fuzzy input variable, it scans the appropriate linked chain of fuzzy output variables that are affected by this change. For each of the fuzzy output variables in the linked chain, a defuzzification process is scheduled which functions to assign a new value to any affected fuzzy output variables (step 414).

In addition, a fuzzy calculation of an output variable occurs (step 414) when the fuzzy output variable is part of an expert system rule (step 412). For example the fuzzy output variable VELOCITY, part of the expert system rule below, would trigger an evaluation when the rule is checked.

1. When (MainTimer is DONE) AND (Evaluation {VELOCITY}>70) Then Turn On Actuator A

In this case, the user has specifically requested to evaluate the value of the fuzzy output variable VELOCITY.

With reference to Figure 22, the forth type of output generated by the compiler is the universe of discourse precalculation data 388. The precalculation of the universe of discourse quantitization is optional but it improves the performance of the control application if performed. Part of the fuzzy evaluation process involves the computation of the fuzzy variable universe of discourse, i.e., the range of all possible values, utilizing the fuzzy membership functions. The data is calculated in accordance with specified quantization of the particular range in combination with the fuzzy variable membership functions. The precalculation data are performed by the compiler and the results are stored in memory assessable by the embedded processor. The precalculation may also be stored in a database accessible to the embedded processor 354 (Figure 18).

The fuzzy evaluation process uses the precalculated values of the fuzzy variables so as to save significant processing time. If the universe of discourse data is not precalculated, the embedded processor would be required to perform the calculations, with the possible results that real time response time, e.g., milliseconds, could not be provided.

The fuzzy logic for control applications and methodologies, including linear, nonlinear and adaptive systems, is described in more detail in A Course in Fuzzy

Systems and Control. Wang, Li-Xin, Prentice Hall, 1997, chapters 16 and 26, incorporated herein by reference.

A key feature of the present invention is the ability of the embedded processors that execute the control application to communicate with each other via a communication network thus effecting a distributed control logic system. A high level block diagram illustrating the use of agents to distribute both expert rule based system processing and fuzzy logic subsystem processing among other OpenBus controller nodes is shown in Figure 26. Two or more OpenBus node controllers 420, each executing a control application and receiving input from sensors and generating output to actuators, can communicate via the network 18 which may comprise any suitable communication means such as a LAN, WAN or even the Internet. The embedded processor within each node controller communicates with other node controllers using what are known as agents.

The passing of agents between node controllers enables the control application logic to be distributed over a plurality of node controllers. Each node controller comprises a communication transport layer which provides the communication and network services to the control application. The communication transport layer may comprise any standard communication protocol or stack that is commercially available, such as TCP/IP, IPX/SPX, etc. The physical and link layers may comprise any standard communications such as Ethernet, Token Ring, FDDI, etc.

The ability to send and receive agents to and from each node controller, enables, for example, a fuzzy evaluation to be performed on one networked embedded processor and the crisp output value to be sent to a second networked embedded processor. In this fashion, a fuzzy control application can be establish over a cluster of standard embedded processors, with the connectivity of all embedded processors to each other provided via the communications network. Further, each embedded processor performs its localized tasks while communicating with other embedded processors to create a fully distributed control logic application.

A flow diagram illustrating the agent handling method of the present invention is shown in Figure 27. During normal operation of the system, each node controller performs the tasks making up the control application (step 430). Event handling is a

major task performed by each node controller. One of the events checked for includes scanning for agents that have arrived from other node controllers (step 432). If the arrival of agent is detected (step 434) the agent is immediately activated (step 436). Once activated, the contents of the agent are processed (step 438).

Further, any of the embedded processors in a node controller can request another networked node controller to perform a fuzzy evaluation to send the crisp output results back to it over the communications network. To accomplish this, the requesting node controller sends a suitable agent to the node controller that is to perform the fuzzy evaluation. The agent sent to the node controller contains the crisp input generating by the requesting node controller. The node controller that performs the fuzzy evaluation functions to package the crisp output results of the fuzzy evaluation in an agent which is sent back to the requesting node controller over the communications network.

A flow diagram illustrating the agent processing method performed when the arrival of an agent has been detected is shown in Figure 28. After the arrival of an agent has been detected by the receiving node controller, the crisp input is extracted from the agent (step 440). The fuzzy logic subsystem within the node controller fuzzifies the crisp input (442) to generate a fuzzy input before performing a fuzzy evaluation (step 444). The results of the fuzzy evaluation are then defuzzified to yield crisp output (step 446). Finally, the crisp output is placed in an agent and transmitted over the network to the requesting node controller (step 448).

As discussed in connection with Figure 27, after an agent is received at a node controller, it is activated (step 436). The agent activation process will now be described in more detail. A block diagram illustrating the passing of agents from one node controller to another is shown in Figure 29. Two node controllers are shown: node controller #1, referenced 450, and node controller #2, referenced 454. Node controller #1 is coupled to one or more sensors and/or actuators 452 and node controller #2 is coupled to one or more sensors and/or actuators 456. Node controllers 450, 454 are constructed similarly to the node controller 420 shown in Figure 18. For clarity purposes, the node controllers shown in Figure 29 are shown simplified.

In order to activate an agent, the agent must first be created in a node controller. In the example shown in Figure 29, node controller #2 declares one or more agents 458,

labeled agent #1, agent #2, agent #N. One way of implementing agents within a node controller is as a stored procedure that is executed by the embedded processor when it is invoked. An agent can be invoked from an external node controller located at a remote site. Both node controllers are coupled to a communications network (not shown for clarity sake) to permit communications therebetween.

When node controller #1 wants to invoke an agent located on node controller #2, it transmits over the network a request to node controller #2 to activate a particular agent. Along with the request, node controller #1 transmits one or more parameters (P1, P2, P3, for example) that originated at node controller #1. Upon receiving the request from node controller #1, node controller #2 activates the appropriate agent in accordance thereto. The agent, once activated, performs the specific tasks and functions it was designed to do and transmits an acknowledgment back to the requesting node controller, i.e., node controller #2. In this manner, the present invention provides distributed communications and processing between a plurality of node controllers making up a distributed control application.

An example of the use of agents to implement a distributed control application will now be presented. A block diagram illustrating an example distributed control application utilizing a control server and an I/O client is shown in Figure 30. Two node controllers are shown: node controller #1, referenced 460, and node controller #2, referenced 462. Node controllers 460, 462 are constructed similarly to the node controller 420 shown in Figure 18. For clarity purposes, the node controllers shown in Figure 30 are shown simplified. In addition, both node controllers are coupled to a communications network 464 to permit communications therebetween. Further, node controller #2 is connected to one or more servo motors 466, one or more encoders 468 and possibly other sensors and/or actuators 470.

Using the present invention, a node controller can be implemented integral with a personal computer as described hereinabove. In this example, node controller #1 is implemented on a PC which serves as a generic operating system. Numerous different control applications can be executed on the node controller, for example, applications involving motion, such as machines and robots. At the same time, the PC permits full integration with other process control elements. The system enables closed loop motion

control via a network 464, wherein the network may comprise LAN/WAN/Internet. The network may also comprise a standard control bus such as Profibus, Interbus, etc.

A system such as shown in Figure 30, enables closed loop motion control via a network, e.g. control bus, without the need for specialized and expensive motion control PC boards. Node controller #1 functions as the control application server, containing the motion and process control portion of the control application. The motion and process control on the server are implemented by defining steps, rules and actions which are translated to p-code or machine code executed on the server (node controller #1). The motors and/or servo motors 466 making up the machine or robot are connected to node controller #2 which functions as an I/O client. In addition, the encoders 468 and other sensors and actuators 470 are connected to the I/O client.

To facilitate complex motion, the I/O client node controller #2 implements a high speed counter which counts the actual motion pulses. The count data, encoder and other sensor data is input by the I/O client (node controller #2) and transmitted to the control application server (node controller #1) via one or more agents. The input data received via the agents are processed and, in response, output commands for driving the servo motors are generated and transmitted back to the I/O client also via agents.

In this fashion, a motion trajectory can be executed over the control network between the control application server and servo motors connected to the I/O client. The encoder position data, sent over the network, is also utilized by the control application on the server to accurately control the motion of the machine or robot. Utilizing relatively simple commands as expressed in the rules and actions, the compiler, generated code to implement dynamic trajectory tracking.

While the invention has been described with respect to a limited number of embodiments, it will be appreciated that many variations, modifications and other applications of the invention may be made.

What is claimed is:

1. A control automation system for controlling a plurality of input and output (I/O) devices in accordance with a control application, said system connected to a network for communicating control automation information, said system comprising:

a development system optionally coupled to said network, said development system generating p-code embodying event triggers, event actions, program logic and fuzzy rules, variables and membership functions implementing said control application; and

at least one node controller coupled to said network for executing in real-time said p-code generated by said development system and for implementing a fuzzy logic inference engine.

2. The system according to claim 1, wherein said node controller comprises:

processor means for managing and controlling the operation of said node controller, said processor means for executing a real-time kernel, said kernel implementing said control application embodied in p-code and implementing said fuzzy logic inference engine;

network interface means for connecting said node controller to said network;

I/O device interface means for connecting said node controller to said plurality of I/O devices; and

bus means for interconnecting together said real-time kernel, said network interface means and said I/O interface means.

3. The system according to claim 1, wherein said node controller comprises:

means for implementing an expert rule based control system; and

means for implementing said fuzzy logic inference engine.

4. The system according to claim 1, wherein said development system comprises a compiler for generating p-code in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of said control application.

5. The system according to claim 1, wherein said development system comprises a compiler for generating machine code in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of said control-application.

6. The system according to claim 1, wherein said development system comprises a compiler for generating code adapted to be executed on math coprocessor means in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of said control application.

7. The system according to claim 1, wherein said development system comprises a compiler for generating code realizing one or more fuzzy evaluation triggers in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of said control application.

8. The system according to claim 1, wherein said development system comprises a compiler for generating code including precalculated data of the fuzzy universe of discourse in accordance with the event triggers, event actions, fuzzy rules, fuzzy variables, fuzzy membership functions and program logic of said control application.

9. The system according to claim 1, wherein said kernel means comprises:

an external input signal scanner for reading, storing and determining changes to external input signals received from said plurality of I/O devices;

an event triggers evaluation module for detecting changes to said external input signals and internal entities, said event triggers evaluation module for determining and resolving all event triggers corresponding to said detected changes;

a scheduler for marking all actions corresponding to said event triggers that resolve true;

an action execution unit for executing and implementing said actions marked for execution by said scheduler;

an entity processor for determining any changes to values assigned to an entity, said entity processor notifying said event triggers evaluation module of said entity value changes; and

means for determining when a fuzzy evaluation is required to be performed.

10. The system according to claim 1, wherein said I/O device interface means comprises a third party I/O interface control board.

11. The system according to claim 1, wherein said bus means comprises a bus contained in but not limited to the group comprising Peripheral Component Interconnect (PCI) bus, VESA Local (VL) bus, V-bus, Industry Standard Architecture (ISA) bus, VersaModule Europa (VME) bus and Extended Industry Standard Architecture (EISA) bus.

12. A node controller apparatus for use in a control automation system, said system for controlling a plurality of input and output (I/O) devices in accordance with a control application, said system including a network for communicating control automation information, said apparatus comprising:

processor means for managing and controlling the operation of said node controller, said processor means for executing a real-time kernel, said kernel implementing said control application embodied in p-code and for implementing a fuzzy logic subsystem;

network interface means for connecting said node controller to said network so as to enable the transfer of one or more agents between node controller connected to said network;

I/O interface means for connecting said node controller to said plurality of I/O devices; and

bus means for interconnecting together said processor means, said kernel means, said network interface means and said I/O interface means.

13. The system according to claim 12, wherein said processor means comprises:

means for implementing an expert rule based control system; and

means for implementing said fuzzy logic inference engine.

14. The system according to claim 12, wherein said processor means comprises:

an expert rule based control system adapted to receive a plurality of input sensor signals from one or more input sensors and adapted to output a plurality of output sensor signals to one or more actuators; and

a fuzzy logic subsystem for implementing a fuzzy logic inference engine, said fuzzy logic subsystem adapted to receive crisp input data from said expert rule based control system, perform fuzzy processing on said input data to yield crisp output data.

15. The system according to claim 14, wherein said fuzzy logic subsystem comprises:

fuzzification means for transforming said crisp input to fuzzy input;

fuzzy logic processing means for performing fuzzy calculations and processing of said fuzzy input and generating fuzzy output in response thereto;

defuzzification means for transforming said fuzzy output into crisp output; and

means for operating said fuzzy logic processing means in accordance with the fuzzy rules, fuzzy variables and associated fuzzy membership functions contained within said control application.

16. The system according to claim 12, wherein said I/O device interface means comprises a third party I/O interface control board.

17. The system according to claim 12, wherein said bus means comprises a bus contained in but not limited to the group comprising Peripheral Component Interconnect (PCI) bus, VESA Local (VL) bus, V-bus, Industry Standard Architecture (ISA) bus, VersaModule Europa (VME) bus and Extended Industry Standard Architecture (EISA) bus.

18. A control automation system for implementing a distributed control application, said system connected to a communication network, said system comprising:

a plurality of node controllers, each node controller comprising:

expert rule based control system means for implementing said control application in accordance with one or more control rules;

fuzzy logic subsystem means for implementing a fuzzy logic inference engine, said fuzzy logic inference engine for evaluating fuzzy variables, expressions and formulas in accordance with one or more fuzzy rules, fuzzy variables and associated membership functions;

network interface means for connecting said node controller to said network so as to enable the transfer of one or more agents between node controllers connected to said network;

I/O interface means for connecting said node controller to a plurality of I/O devices: and

bus means for interconnecting together said expert rule based control system means, said network interface means and said I/O interface means.

19. The system according to claim 18, wherein said fuzzy logic subsystem comprises:

fuzzification means for transforming said crisp input to fuzzy input;

fuzzy logic processing means for performing fuzzy calculations and processing of said fuzzy input and generating fuzzy output in response thereto;

defuzzification means for transforming said fuzzy output into crisp output; and

means for operating said fuzzy logic processing means in accordance with the fuzzy rules, fuzzy variables and associated fuzzy membership functions contained within said control application.

20. In a control automation system connected to a communication network, a method of implementing a distributed control application, said method comprising the steps of:

providing a plurality of node controllers wherein each node controller includes expert rule based control system means for implementing said control application in accordance with one or more control rules and fuzzy logic subsystem means for implementing a fuzzy logic inference engine;

evaluating fuzzy variables, expressions and formulas in accordance with one or more fuzzy rules, fuzzy variables and associated membership functions making up said control application;

connecting said node controller to said network so as to enable the transfer of one or more agents between node controllers connected to said network;

connecting said node controller to a plurality of I/O devices; and

interconnecting together said expert rule based control system means, said network interface means and said I/O interface means.

21. The method according to claim 20, wherein said step of evaluating comprises the steps of:

detecting, in accordance with said control rules, that a fuzzy evaluation is to be performed;

fuzzifying crisp input output by said expert rule based control system means to yield a fuzzy input;

performing a fuzzy evaluation of said fuzzy input in accordance with said fuzzy rules, fuzzy variables and associated membership functions making up said control application to yield a fuzzy output;

defuzzifying said fuzzy output to yield crisp output; and

sending said crisp output results to said expert rule based control system.

22. In a computer system, a method of generating p-code for execution on a node controller as part of a control automation system for controlling a plurality of input and output (I/O) devices in accordance with a user's application, said application including event triggers, event actions and program logic, said method comprising the steps of:

generating a plurality of pointer tables, each pointer table associated with either an external input signal or an entity, each pointer table comprising a plurality of pointer entries, each pointer entry pointing to an event trigger;

generating an event trigger table, said event trigger table comprising a plurality of event trigger entries, each event trigger entry corresponding to an action that references the particular external input signal or entity that points thereto;

generating a plurality of actions, each of said actions comprising at least one frame, said actions, said actions representing the generation of output signals and/or the modification of said internal entities; and

wherein said plurality of pointer tables, said event trigger table and said plurality of actions generated in accordance with said event triggers, event actions and program logic making up said user's application.

23. A kernel for implementation on a computing means, said computing means part of a control automation system for controlling a plurality of input and output (I/O) devices in accordance with a control application, said computing means including an expert rule based control system and a fuzzy logic subsystem, said kernel comprising:

an external input signal scanner for reading, storing and determining changes to external input signals received from said plurality of I/O devices;

an event triggers evaluation module for detecting changes to said external input signals and internal entities, said event triggers evaluation module for determining and resolving all event triggers corresponding to said detected changes;

a scheduler for marking all actions corresponding to said event triggers that resolve true;

an action execution unit for executing and implementing said actions marked for execution by said scheduler;

an entity processor for determining any changes to values assigned to an entity, said entity processor notifying said event triggers evaluation module of said entity value changes;

means for determining when a fuzzy evaluation is required to be performed; and

means for sending crisp input to said fuzzy logic subsystem and receiving crisp output in response thereto.

24. The kernel according to claim 23, wherein said action execution unit performs a method comprising the steps of:

reading the p-code contents of a frame;

analyzing said p-code;

reading the values of external input signals and/or internal entities; and

performing the command embodied in said p-code;

generating any output signals in accordance with said command; and

modifying any entity values in accordance with said command.

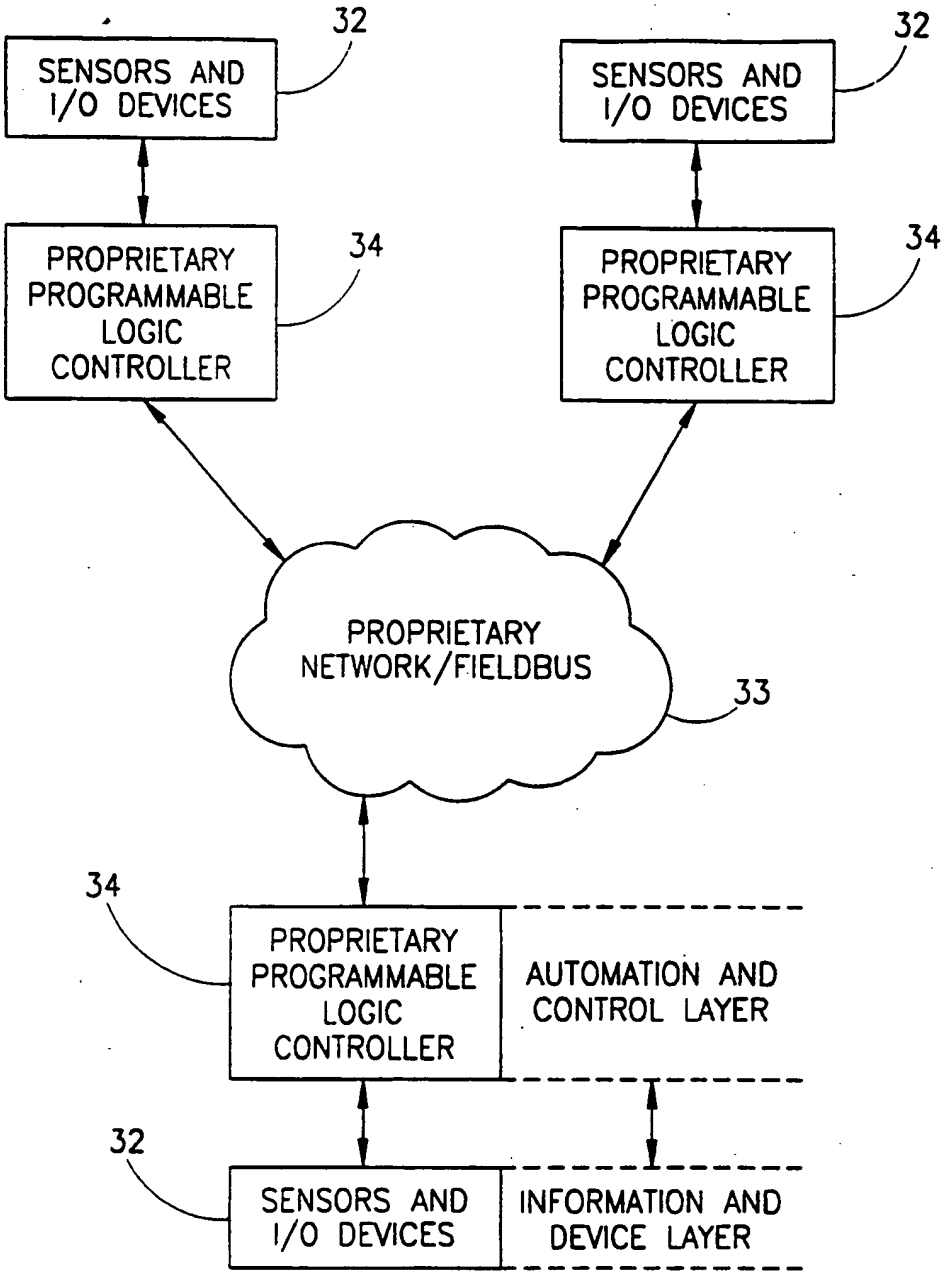



FIG.1
PRIOR ART

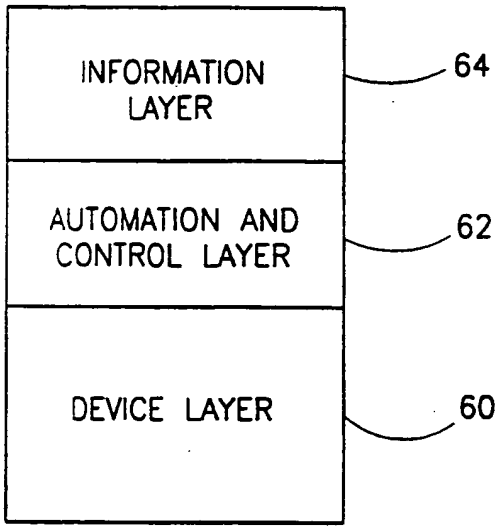

INFORMATION LAYER
64
AUTOMATION AND CONTROL LAYER
62
DEVICE LAYER
60

FIG.2

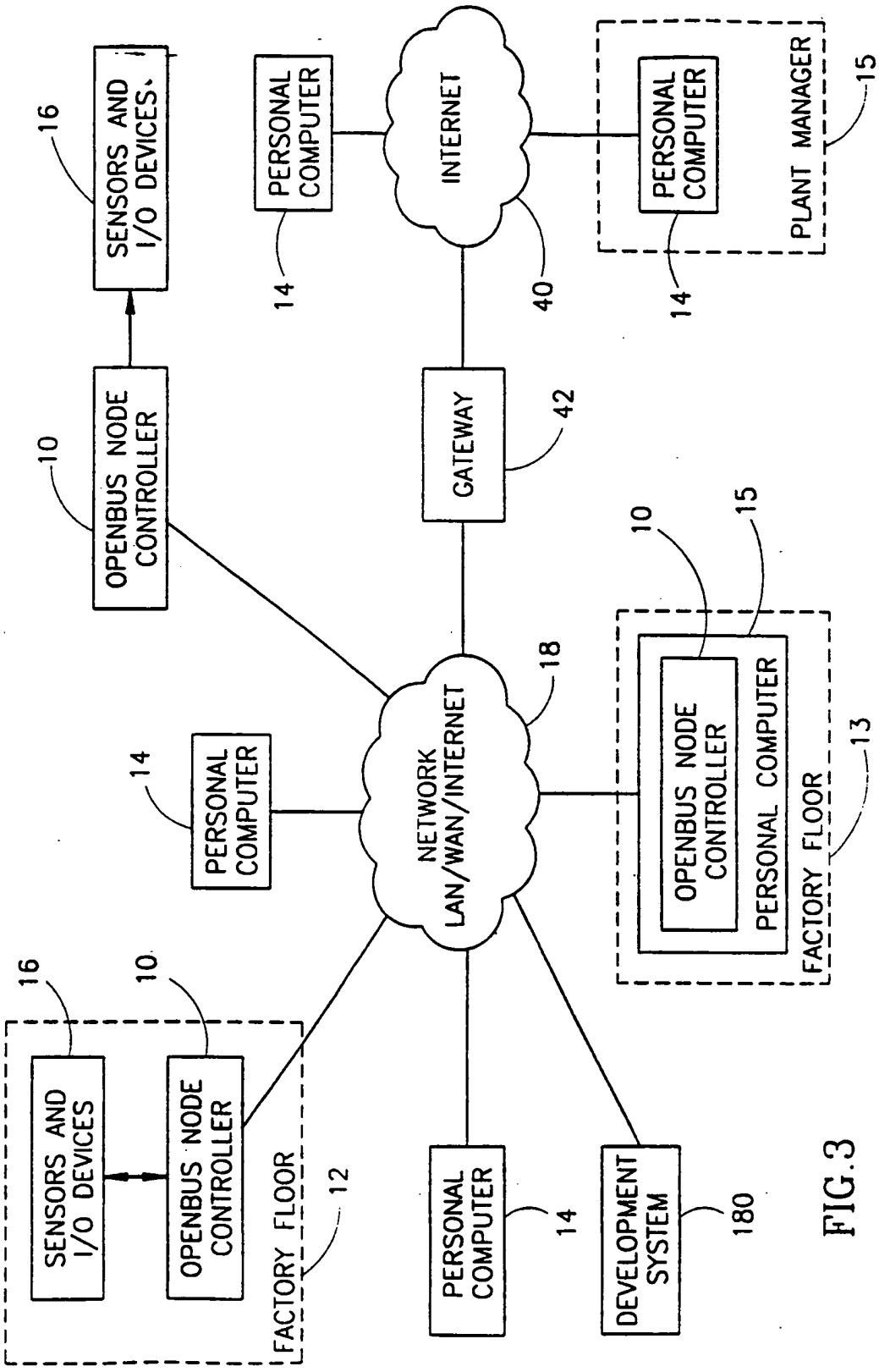

SENSORS AND I/O DEVICES
16
OPENBUS NODE CONTROLLER
10
PERSONAL COMPUTER
14
INTERNET
40
PERSONAL COMPUTER
14
PLANT MANAGER
15
GATEWAY
42
NETWORK LAN/WAN/INTERNET
18
PERSONAL COMPUTER
14
OPENBUS NODE CONTROLLER
10
PERSONAL COMPUTER
15
FACTORY FLOOR
13
SENSORS AND I/O DEVICES
16
OPENBUS NODE CONTROLLER
10
FACTORY FLOOR
12
PERSONAL COMPUTER
14
DEVELOPMENT SYSTEM
180

FIG.3

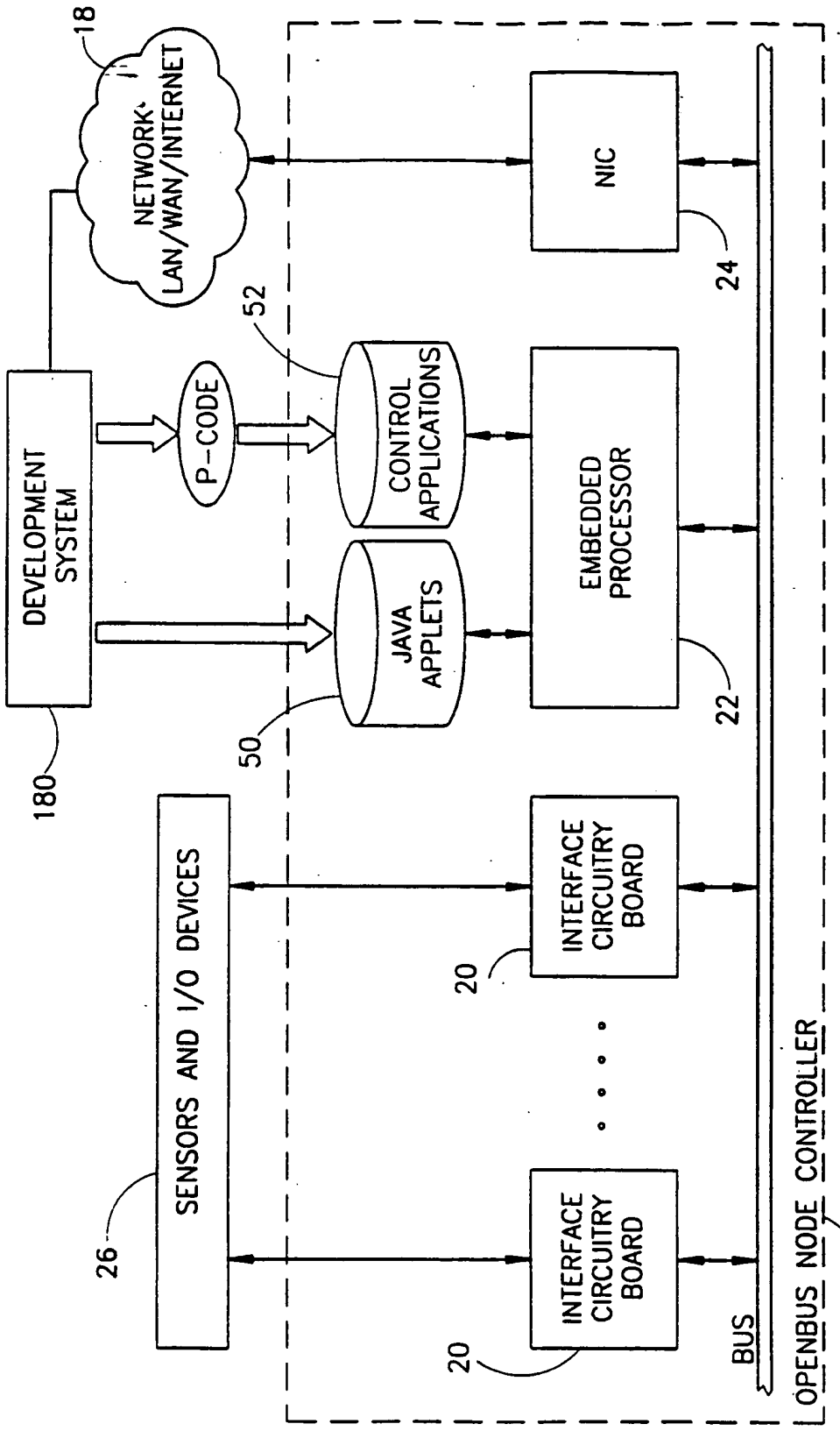

18
NETWORK
LAN/WAN/INTERNET
NIC
24
52
CONTROL APPLICATIONS
P-CODE
DEVELOPMENT SYSTEM
EMBEDDED PROCESSOR
JAVA APPLETS
50
22
180
SENSORS AND I/O DEVICES
INTERFACE CIRCUITRY BOARD
20
INTERFACE CIRCUITRY BOARD
20
26
BUS
OPENBUS NODE CONTROLLER
10

FIG.4

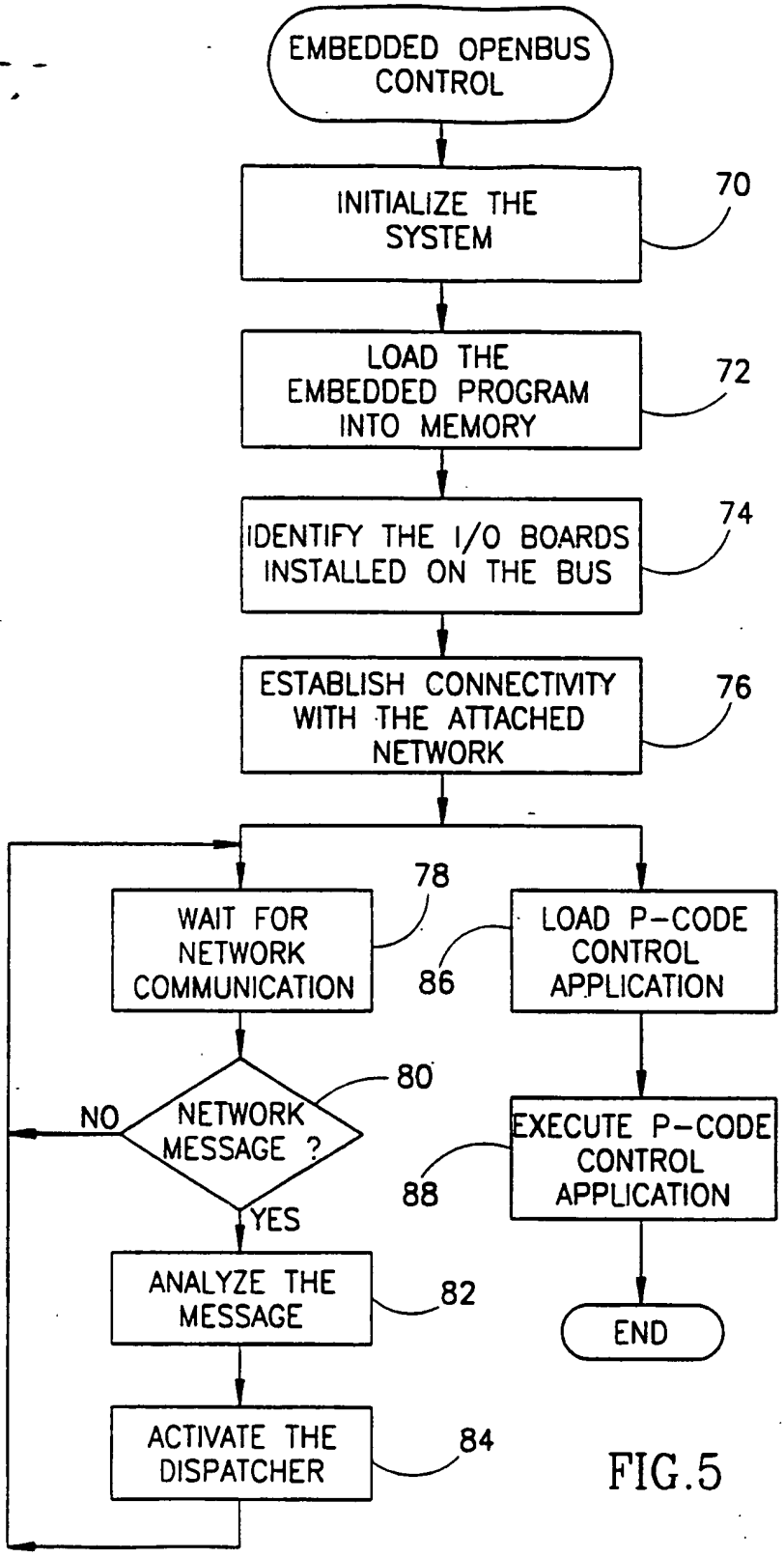

EMBEDDED OPENBUS CONTROL
INITIALIZE THE SYSTEM
70
LOAD THE EMBEDDED PROGRAM INTO MEMORY
72
IDENTIFY THE I/O BOARDS INSTALLED ON THE BUS
74
ESTABLISH CONNECTIVITY WITH THE ATTACHED NETWORK
76
WAIT FOR NETWORK COMMUNICATION
78
LOAD P-CODE CONTROL APPLICATION
86
NETWORK MESSAGE ?
80
NO
YES
EXECUTE P-CODE CONTROL APPLICATION
88
ANALYZE THE MESSAGE
82
END
ACTIVATE THE DISPATCHER
84

FIG.5

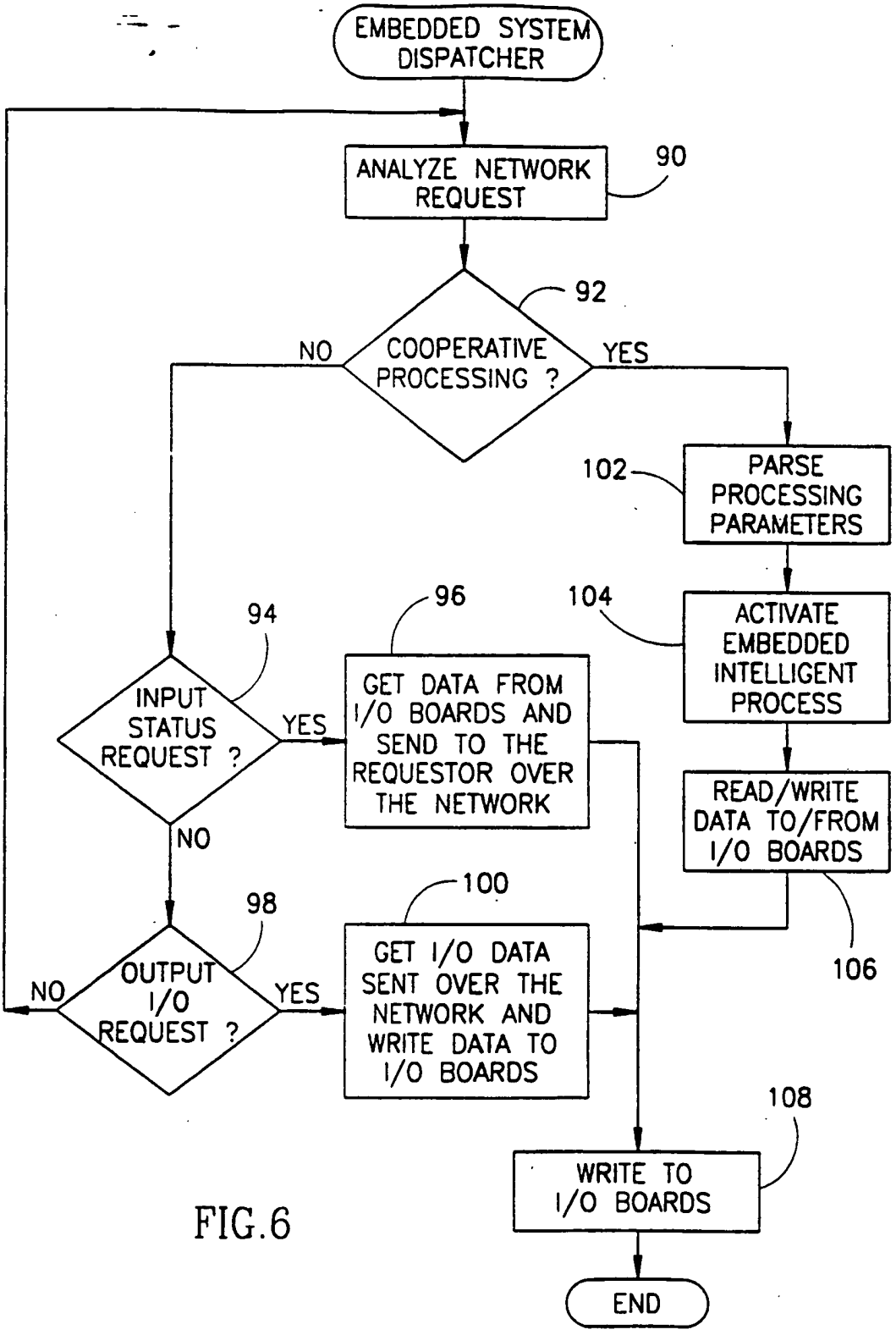

EMBEDDED SYSTEM DISPATCHER
ANALYZE NETWORK REQUEST
90
COOPERATIVE PROCESSING ?
92
NO
YES
PARSE PROCESSING PARAMETERS
102
ACTIVATE EMBEDDED INTELLIGENT PROCESS
104
READ/WRITE DATA TO/FROM I/O BOARDS
106
INPUT STATUS REQUEST ?
94
YES
NO
GET DATA FROM I/O BOARDS AND SEND TO THE REQUESTOR OVER THE NETWORK
96
OUTPUT I/O REQUEST ?
98
NO
YES
GET I/O DATA SENT OVER THE NETWORK AND WRITE DATA TO I/O BOARDS
100
WRITE TO I/O BOARDS
108
END

FIG.6

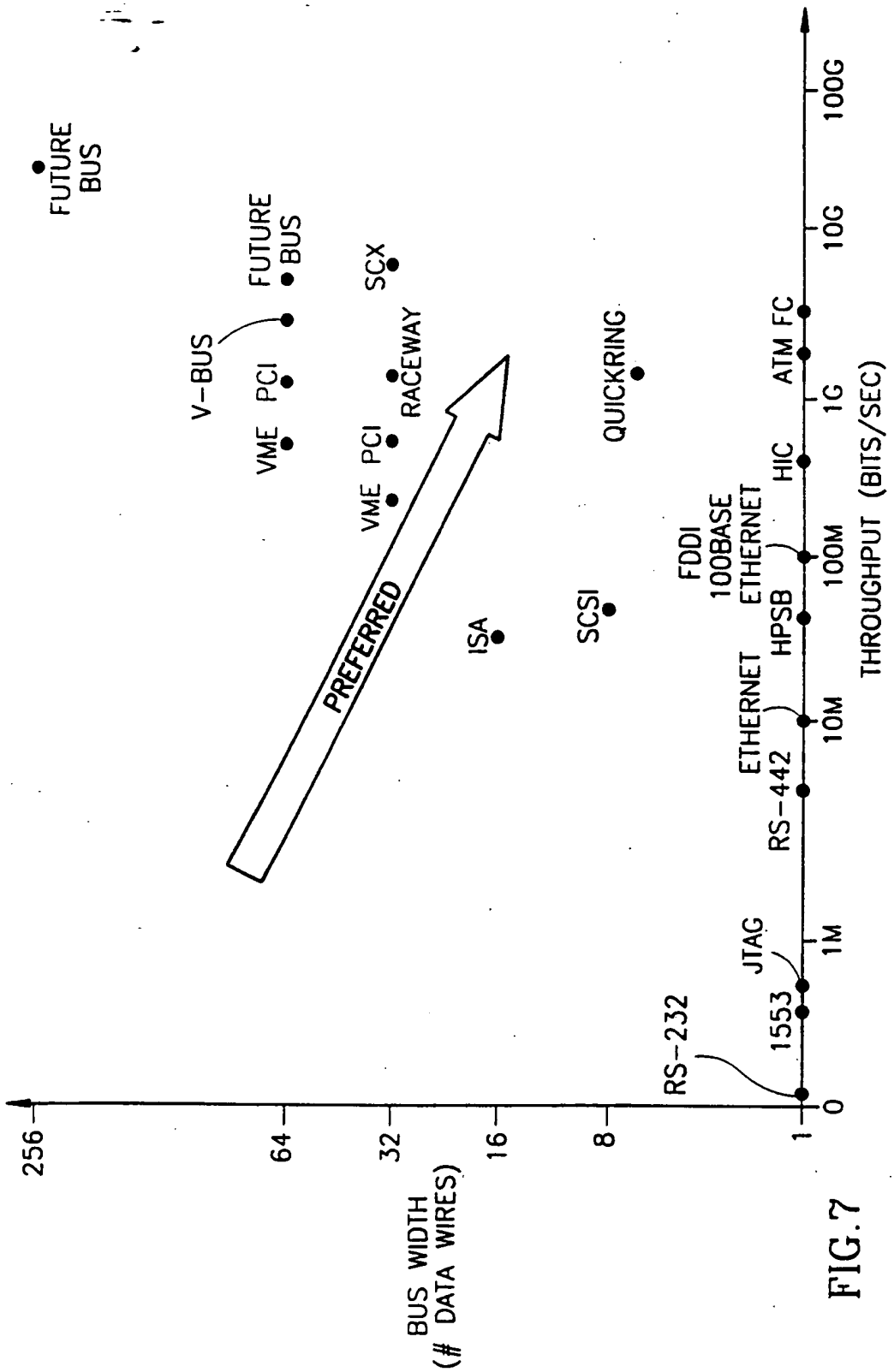

FUTURE BUS
V-BUS
FUTURE BUS
VME
PCI
SCX
RACEWAY
VME
PCI
PREFERRED
ISA
SCSI
QUICKRING
FDDI
100BASE ETHERNET
HIC
ATM FC
HPSB
ETHERNET
RS-442
JTAG
1553
RS-232
256
64
32
16
8
1
0
1M
10M
100M
1G
10G
100G
BUS WIDTH (# DATA WIRES)
THROUGHPUT (BITS/SEC)

FIG.7

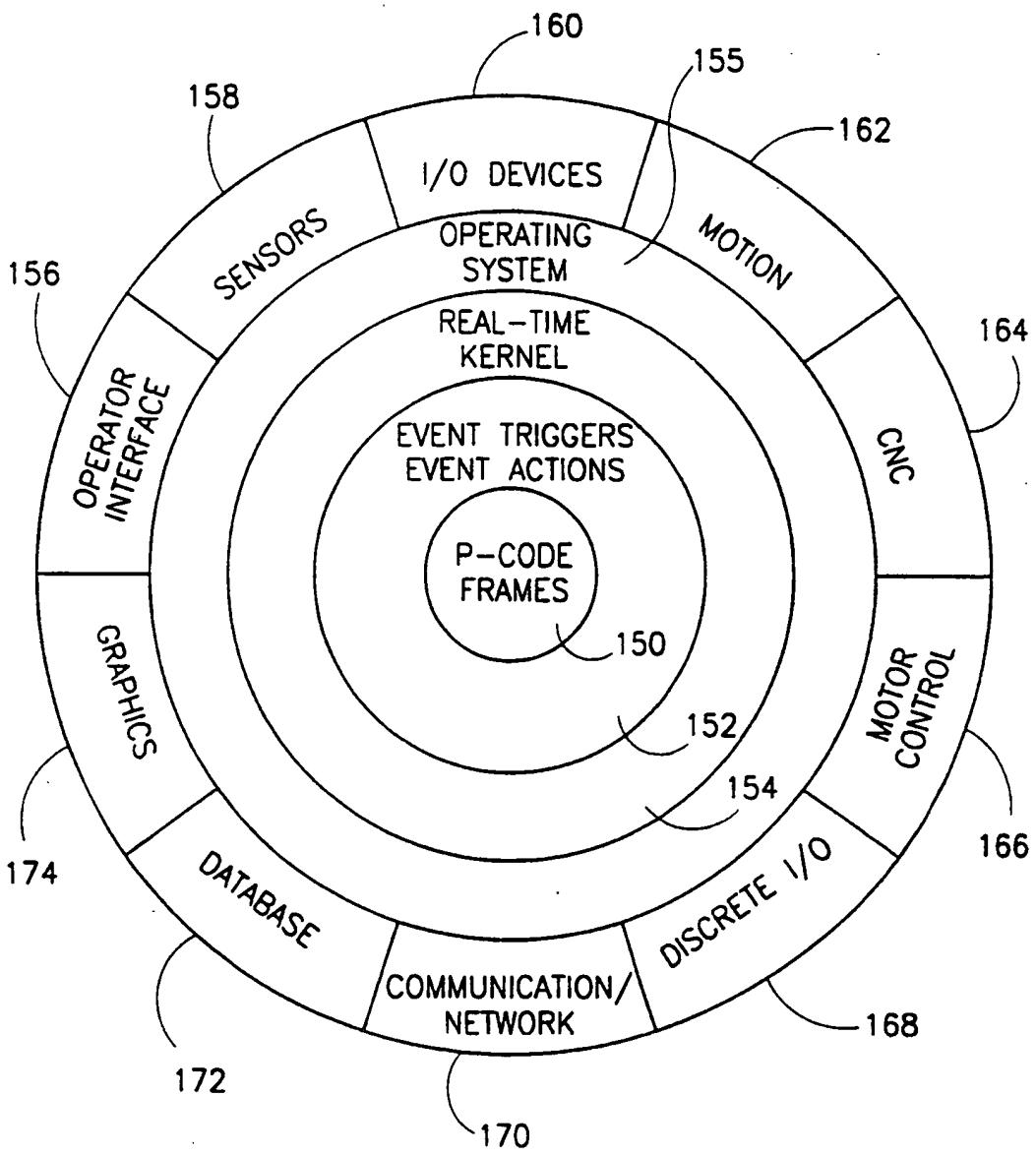

160
155
158
162
I/O DEVICES
SENSORS
OPERATING SYSTEM
MOTION
156
REAL-TIME KERNEL
164
EVENT TRIGGERS
EVENT ACTIONS
OPERATOR INTERFACE
CNC
P-CODE FRAMES
150
152
GRAPHICS
MOTOR CONTROL
154
174
DATABASE
DISCRETE I/O
166
COMMUNICATION/ NETWORK
168
172
170

FIG.8

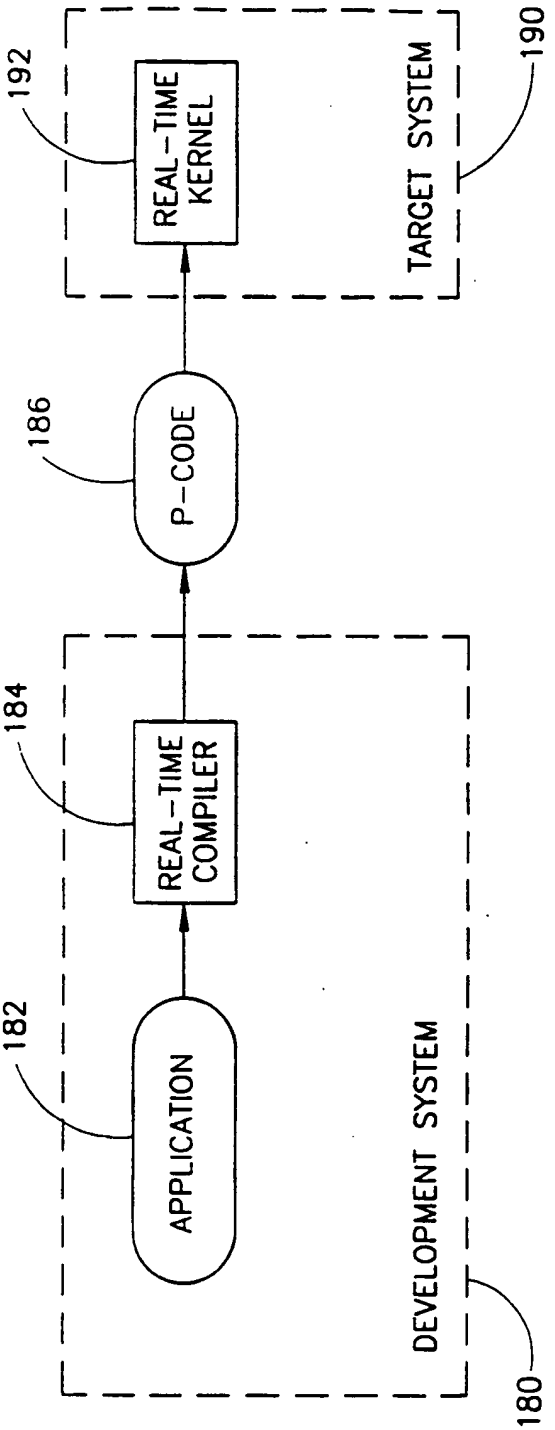

192
REAL-TIME KERNEL
TARGET SYSTEM
190
186
P-CODE
184
REAL-TIME COMPILER
182
APPLICATION
DEVELOPMENT SYSTEM
180

FIG.9

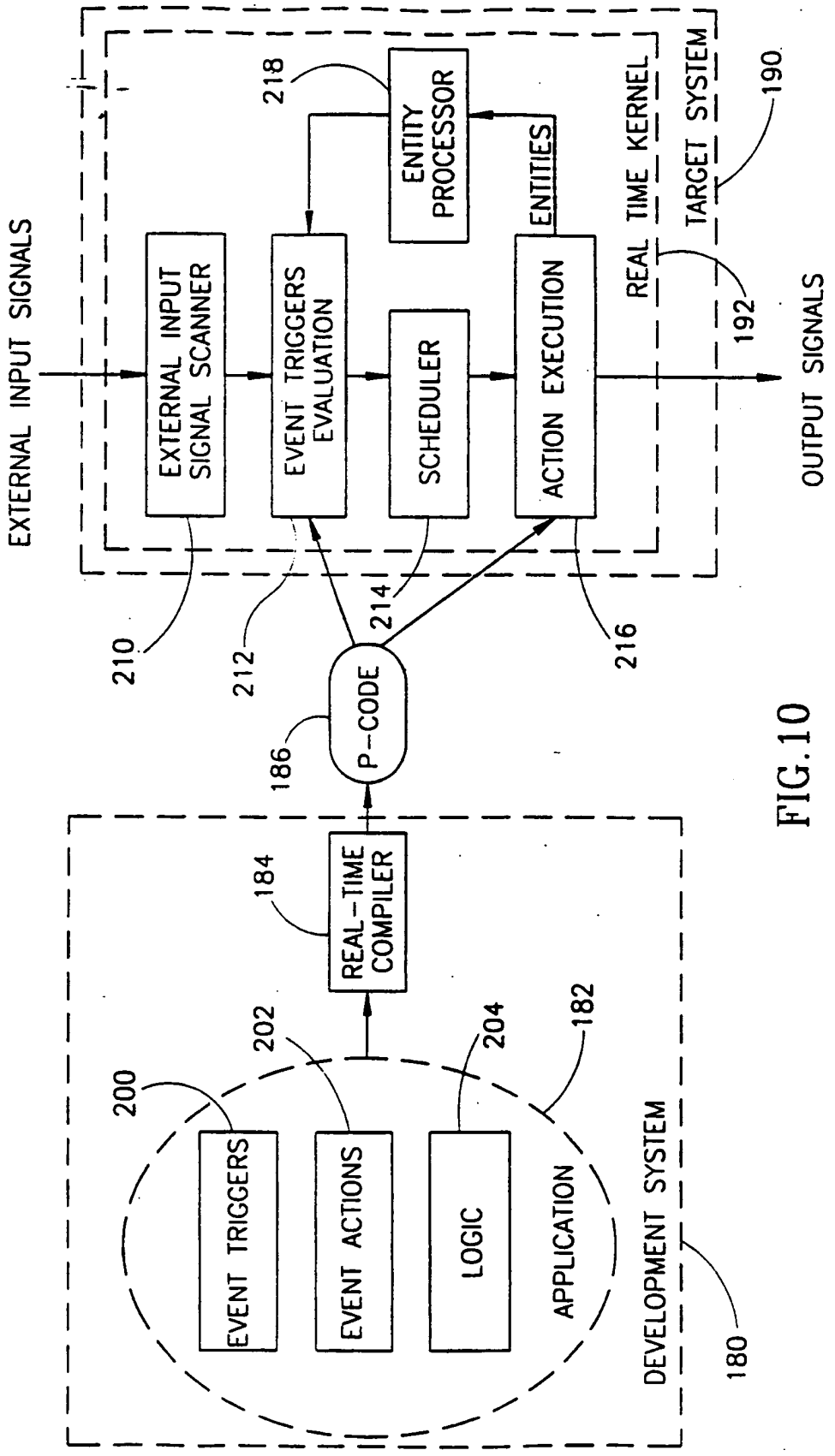

EXTERNAL INPUT SIGNALS
EXTERNAL INPUT SIGNAL SCANNER
EVENT TRIGGERS EVALUATION
SCHEDULER
ACTION EXECUTION
ENTITY PROCESSOR
ENTITIES
REAL TIME KERNEL
TARGET SYSTEM
OUTPUT SIGNALS
190
192
218
210
212
214
216
P-CODE
186
REAL-TIME COMPILER
184
DEVELOPMENT SYSTEM
180
APPLICATION
182
EVENT TRIGGERS
EVENT ACTIONS
LOGIC
200
202
204

FIG.10

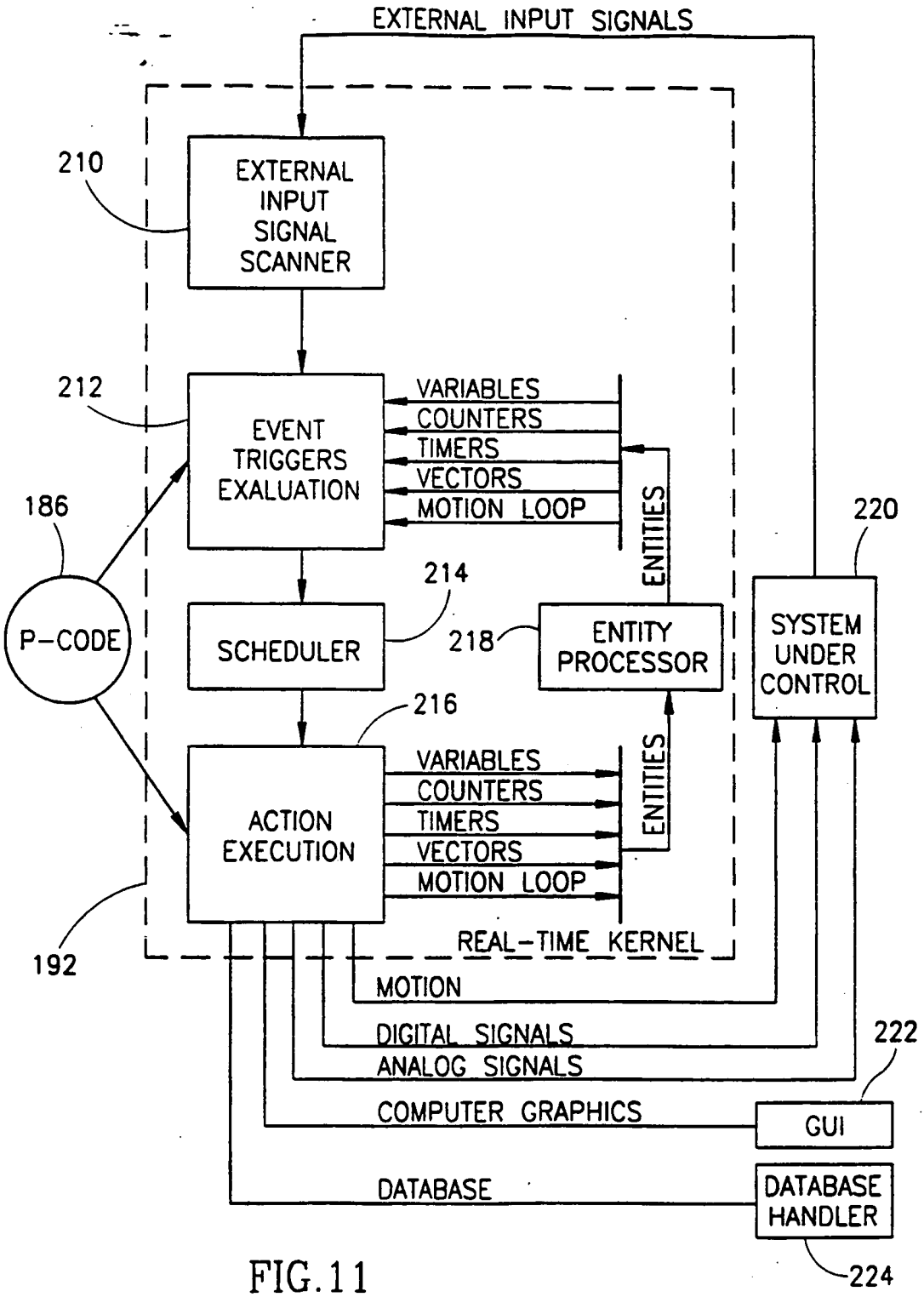

EXTERNAL INPUT SIGNALS
210
EXTERNAL INPUT SIGNAL SCANNER
212
EVENT TRIGGERS EXALUATION
VARIABLES
COUNTERS
TIMERS
VECTORS
MOTION LOOP
ENTITIES
186
P-CODE
214
SCHEDULER
218
ENTITY PROCESSOR
220
SYSTEM UNDER CONTROL
216
ACTION EXECUTION
VARIABLES
COUNTERS
TIMERS
VECTORS
MOTION LOOP
ENTITIES
192
REAL-TIME KERNEL
MOTION
DIGITAL SIGNALS
ANALOG SIGNALS
222
COMPUTER GRAPHICS
GUI
DATABASE
DATABASE HANDLER
224

FIG.11

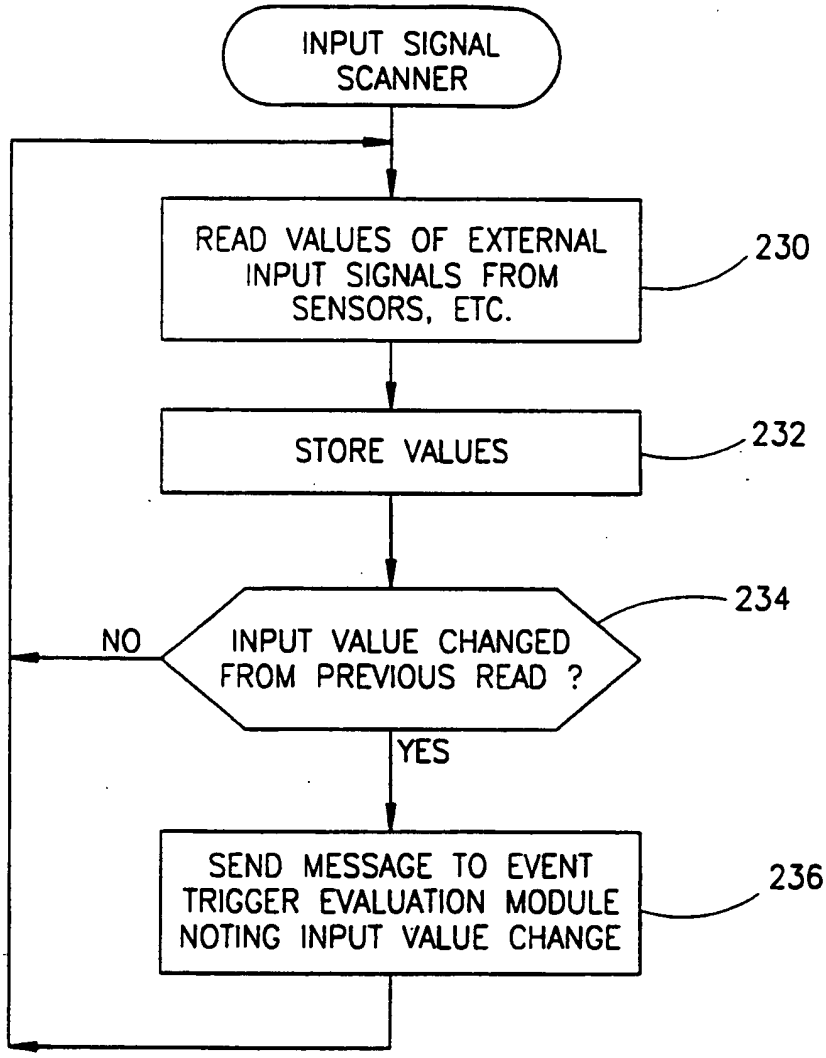

INPUT SIGNAL SCANNER
READ VALUES OF EXTERNAL INPUT SIGNALS FROM SENSORS, ETC.
230
STORE VALUES
232
INPUT VALUE CHANGED FROM PREVIOUS READ ?
234
NO
YES
SEND MESSAGE TO EVENT TRIGGER EVALUATION MODULE NOTING INPUT VALUE CHANGE
236

FIG.12

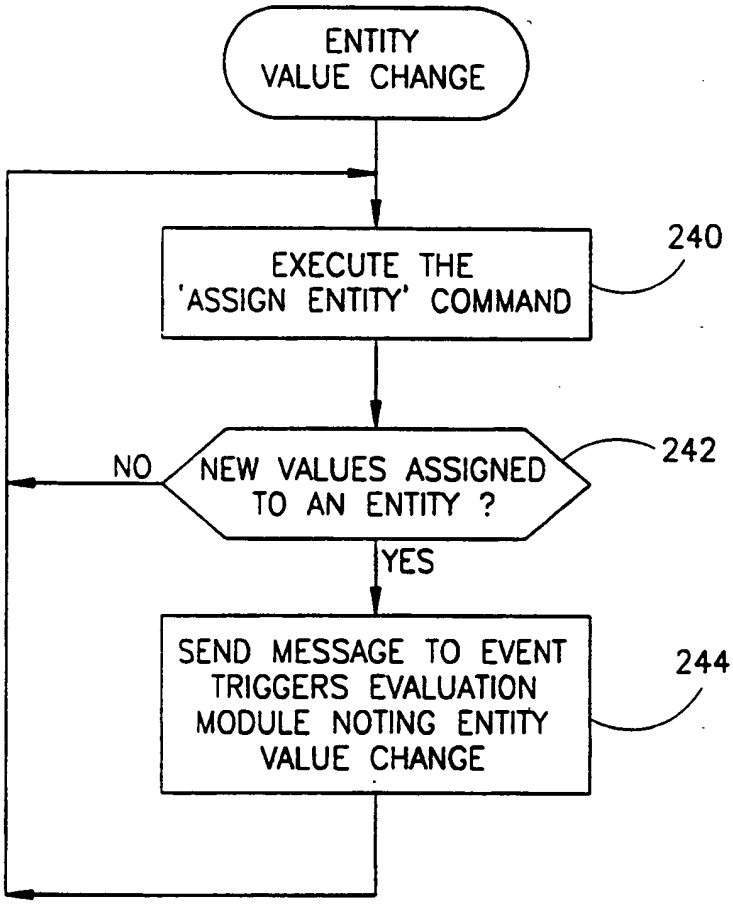

ENTITY VALUE CHANGE
EXECUTE THE 'ASSIGN ENTITY' COMMAND
240
NEW VALUES ASSIGNED TO AN ENTITY ?
242
NO
YES
SEND MESSAGE TO EVENT TRIGGERS EVALUATION MODULE NOTING ENTITY VALUE CHANGE
244

FIG.13

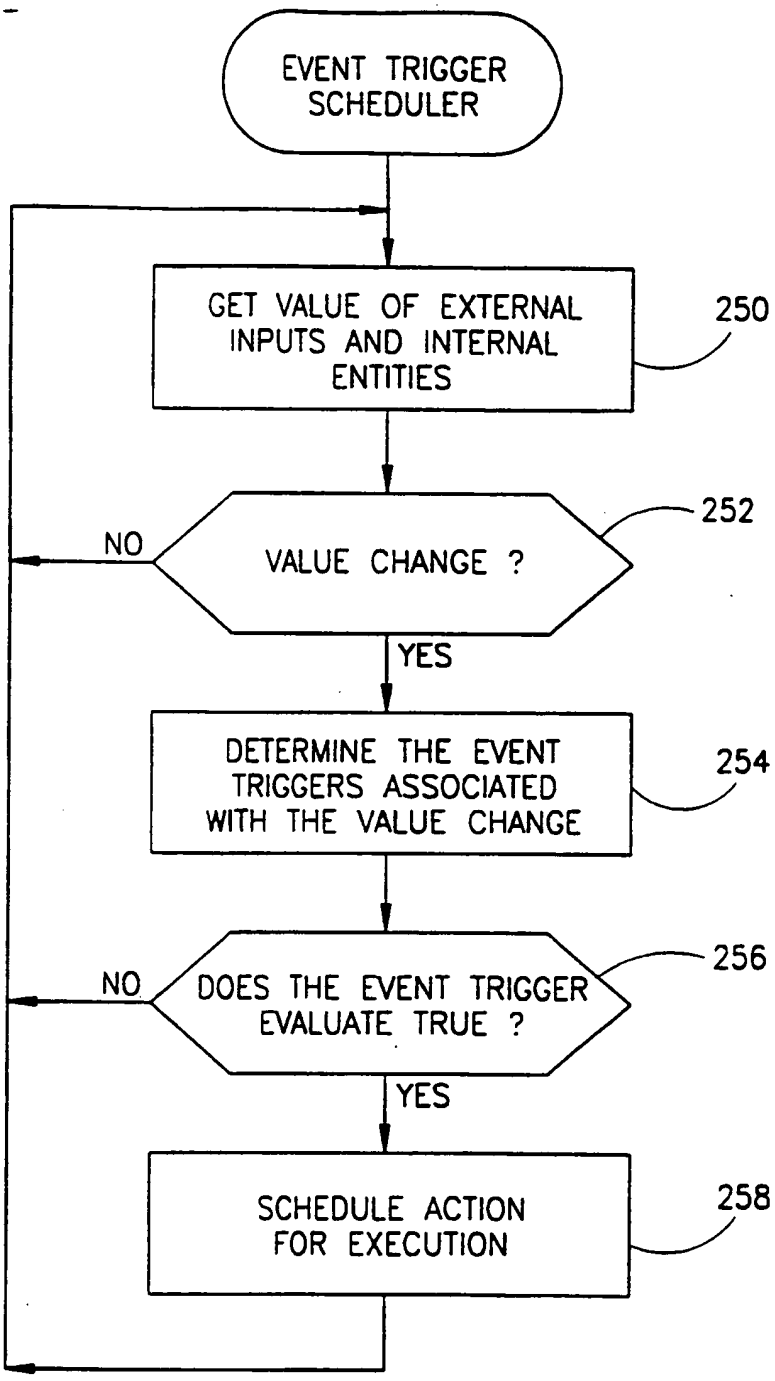

EVENT TRIGGER SCHEDULER
GET VALUE OF EXTERNAL INPUTS AND INTERNAL ENTITIES
250
VALUE CHANGE ?
252
NO
YES
DETERMINE THE EVENT TRIGGERS ASSOCIATED WITH THE VALUE CHANGE
254
DOES THE EVENT TRIGGER EVALUATE TRUE ?
256
NO
YES
SCHEDULE ACTION FOR EXECUTION
258

FIG.14

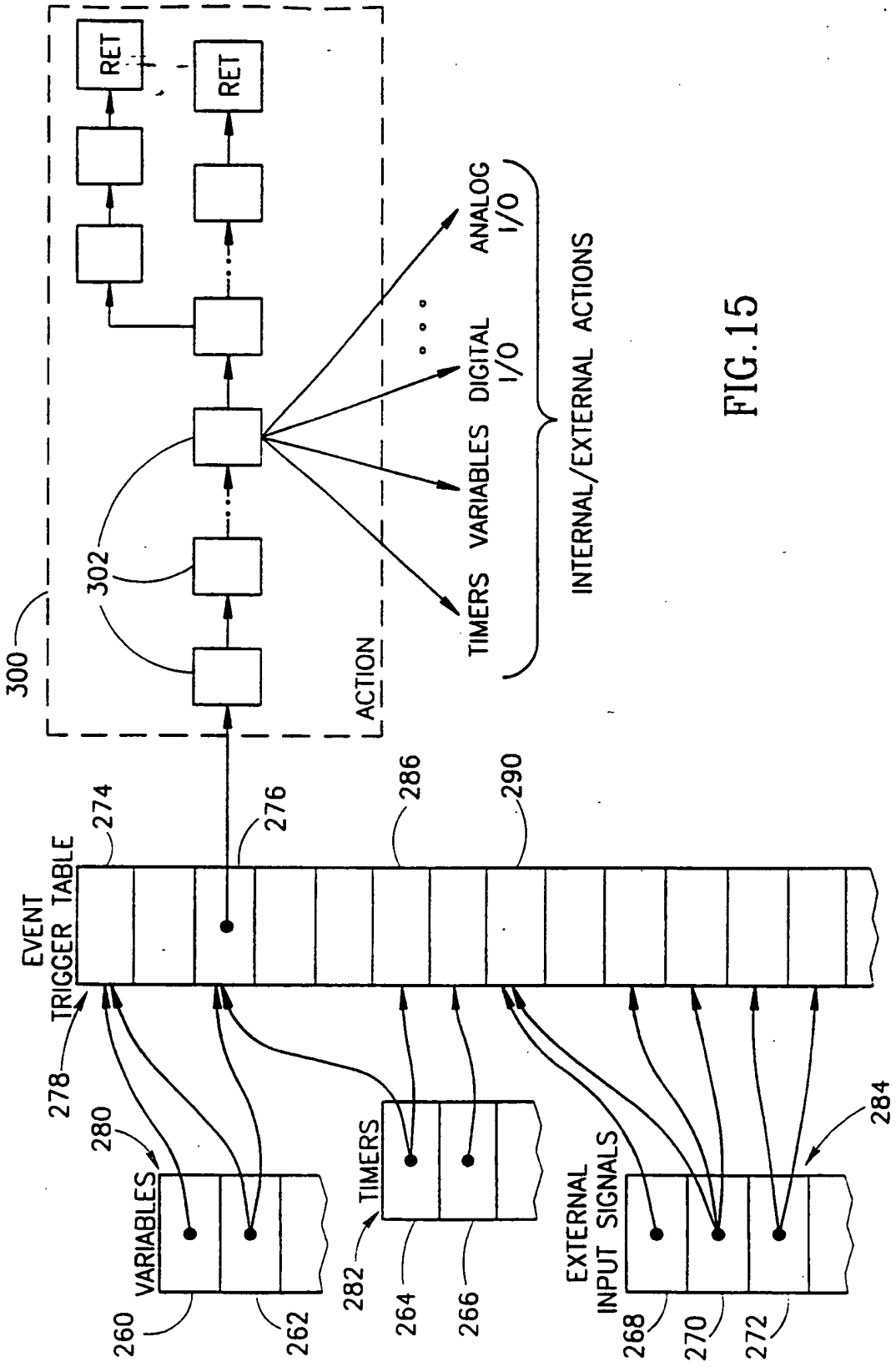

RET
RET
300
302
ACTION
TIMERS
VARIABLES
DIGITAL I/O
ANALOG I/O
INTERNAL/EXTERNAL ACTIONS
EVENT TRIGGER TABLE
274
276
286
290
278
280
VARIABLES
TIMERS
EXTERNAL INPUT SIGNALS
284
260
262
282
264
266
268
270
272

FIG.15

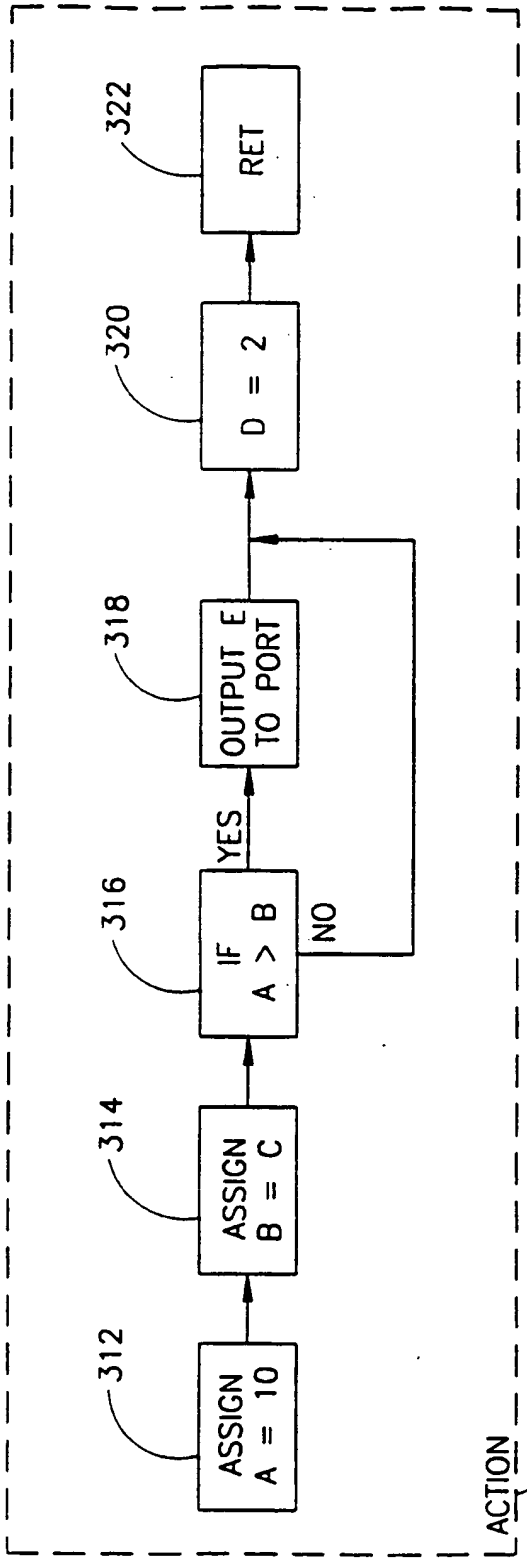

ACTION
310
ASSIGN A = 10
312
ASSIGN B = C
314
IF A > B
316
YES
NO
OUTPUT E TO PORT
318
D = 2
320
RET
322

FIG.16

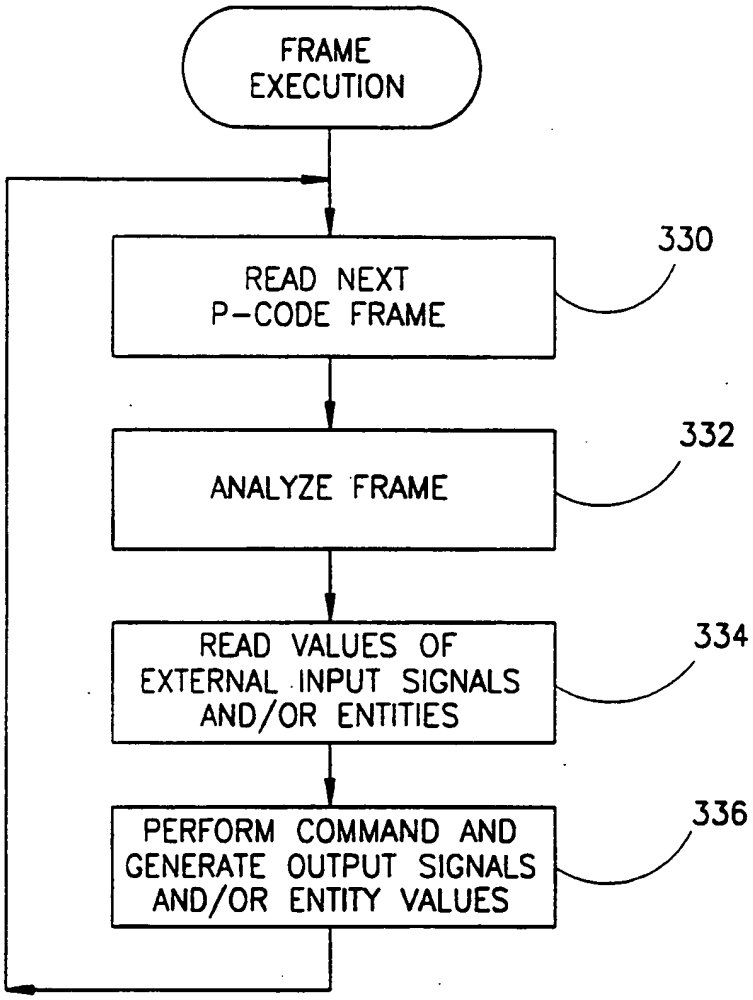

FRAME EXECUTION
READ NEXT P-CODE FRAME
330
ANALYZE FRAME
332
READ VALUES OF EXTERNAL INPUT SIGNALS AND/OR ENTITIES
334
PERFORM COMMAND AND GENERATE OUTPUT SIGNALS AND/OR ENTITY VALUES
336

FIG.17

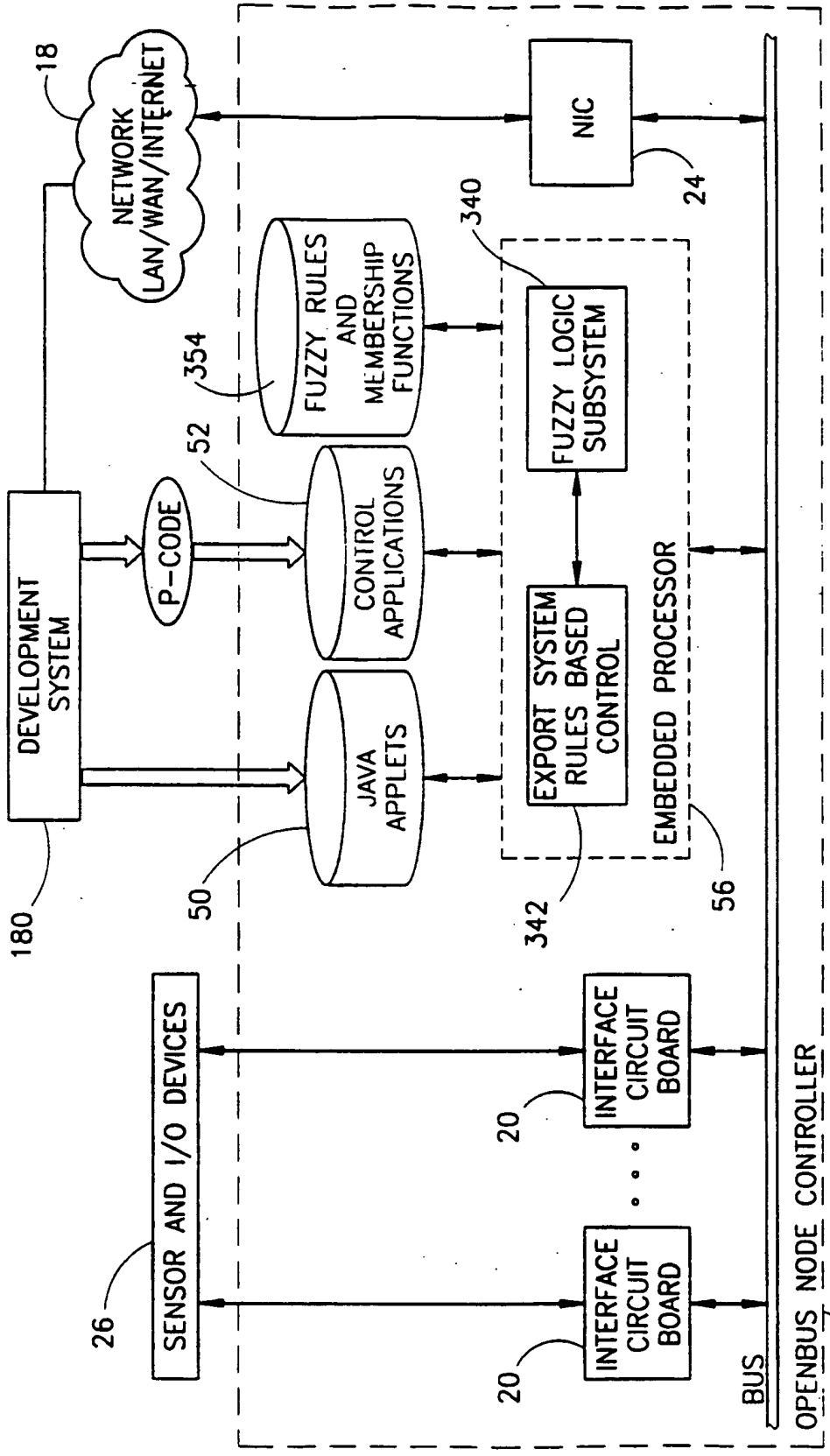

18
NETWORK LAN/WAN/INTERNET
NIC
24
340
FUZZY RULES AND MEMBERSHIP FUNCTIONS
354
FUZZY LOGIC SUBSYSTEM
52
P-CODE
CONTROL APPLICATIONS
DEVELOPMENT SYSTEM
JAVA APPLETS
EXPORT SYSTEM RULES BASED CONTROL
EMBEDDED PROCESSOR
180
50
342
56
SENSOR AND I/O DEVICES
INTERFACE CIRCUIT BOARD
20
INTERFACE CIRCUIT BOARD
20
26
BUS
OPENBUS NODE CONTROLLER
420

FIG.18

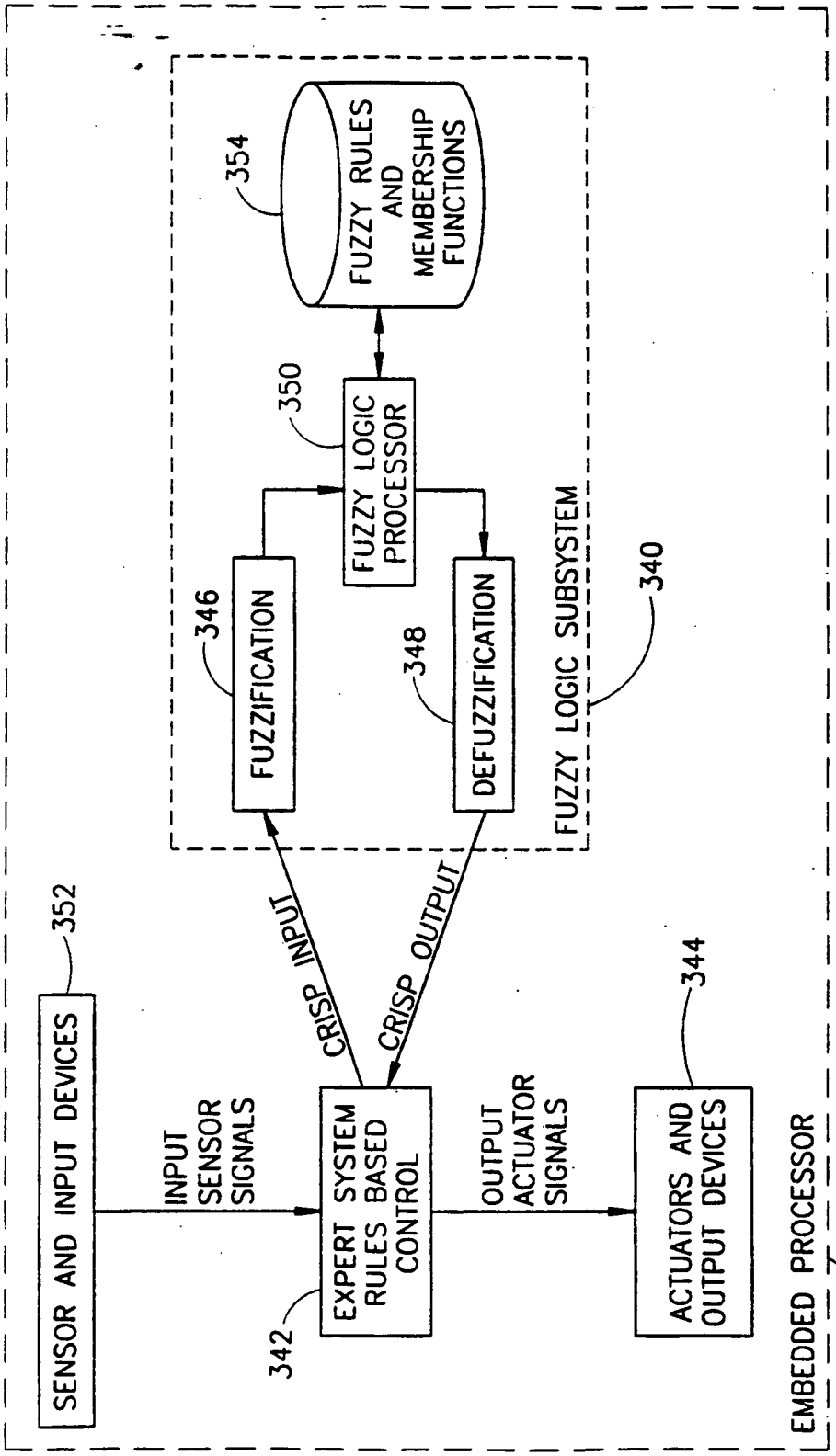

EMBEDDED PROCESSOR
56
SENSOR AND INPUT DEVICES
352
INPUT SENSOR SIGNALS
EXPERT SYSTEM RULES BASED CONTROL
342
OUTPUT ACTUATOR SIGNALS
ACTUATORS AND OUTPUT DEVICES
344
CRISP INPUT
CRISP OUTPUT
FUZZIFICATION
346
DEFUZZIFICATION
348
FUZZY LOGIC PROCESSOR
350
FUZZY RULES AND MEMBERSHIP FUNCTIONS
354
FUZZY LOGIC SUBSYSTEM
340

FIG.19

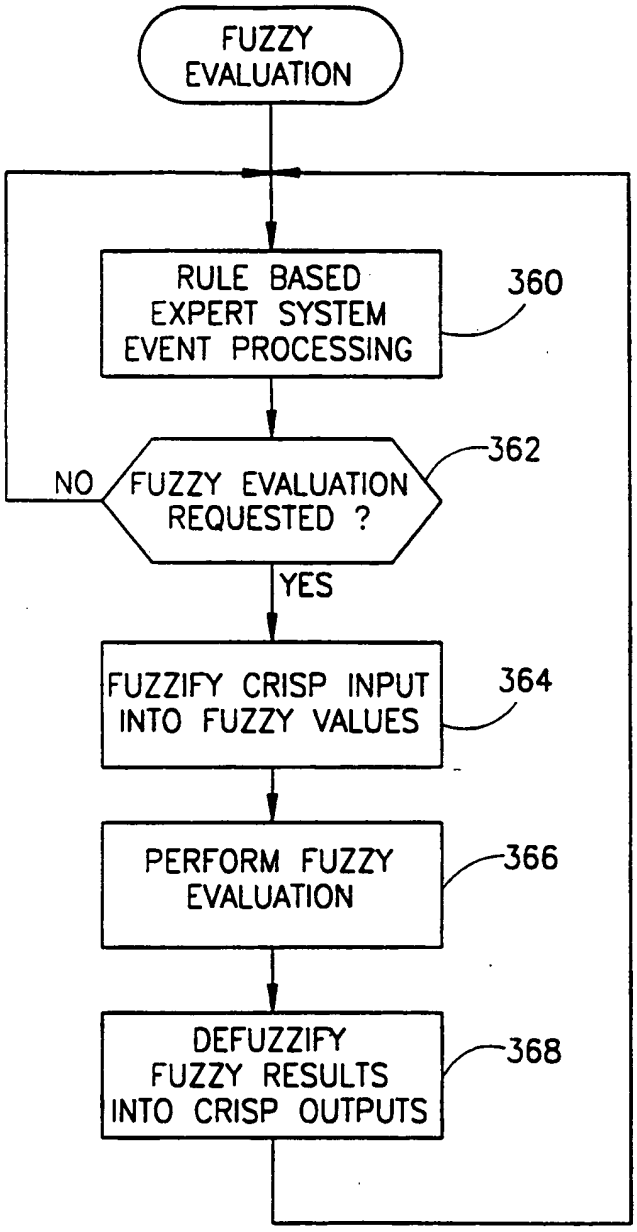

FUZZY EVALUATION
RULE BASED EXPERT SYSTEM EVENT PROCESSING
360
NO
FUZZY EVALUATION REQUESTED ?
362
YES
FUZZIFY CRISP INPUT INTO FUZZY VALUES
364
PERFORM FUZZY EVALUATION
366
DEFUZZIFY FUZZY RESULTS INTO CRISP OUTPUTS
368

FIG.20

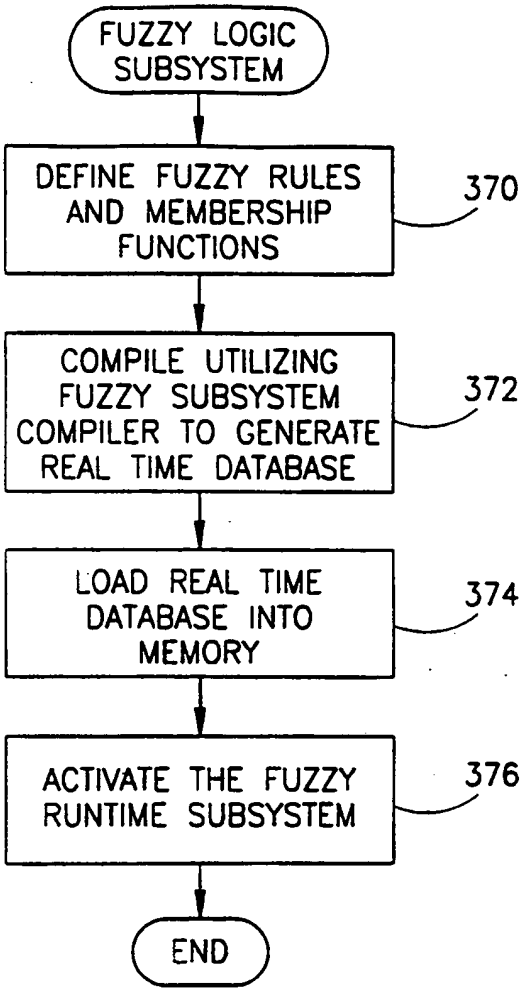

FUZZY LOGIC SUBSYSTEM
DEFINE FUZZY RULES AND MEMBERSHIP FUNCTIONS
370
COMPILE UTILIZING FUZZY SUBSYSTEM COMPILER TO GENERATE REAL TIME DATABASE
372
LOAD REAL TIME DATABASE INTO MEMORY
374
ACTIVATE THE FUZZY RUNTIME SUBSYSTEM
376
END

FIG.21

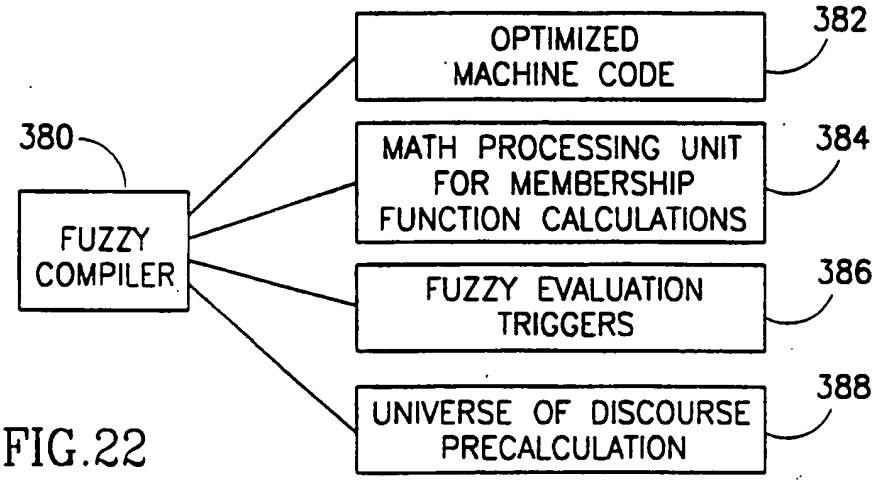

380
FUZZY COMPILER
OPTIMIZED MACHINE CODE
382
MATH PROCESSING UNIT FOR MEMBERSHIP FUNCTION CALCULATIONS
384
FUZZY EVALUATION TRIGGERS
386
UNIVERSE OF DISCOURSE PRECALCULATION
388

FIG.22

FIG.23A

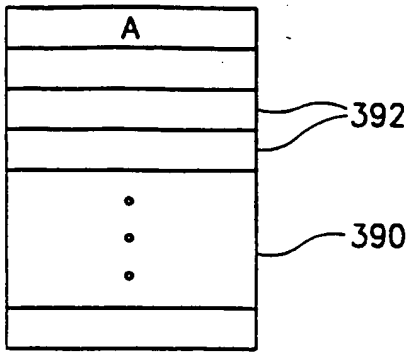



FIG.23B

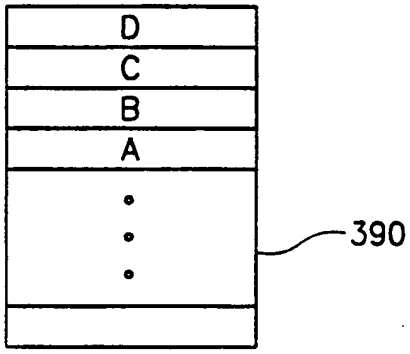



FIG.23C

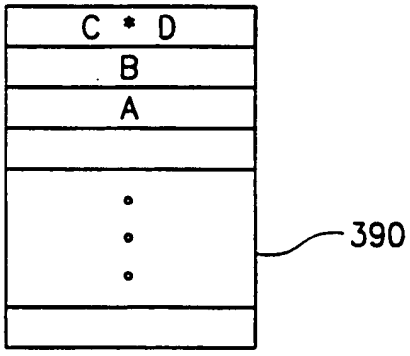


FIG.23D

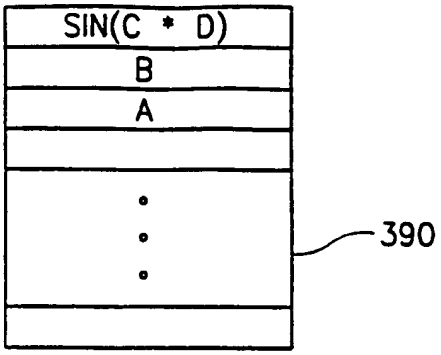



FIG.23E

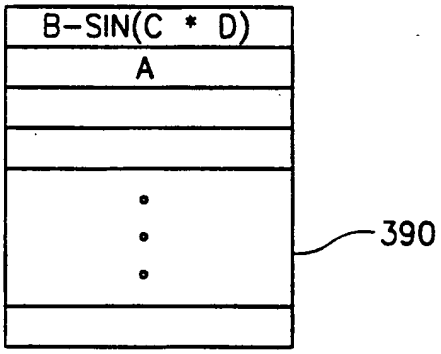
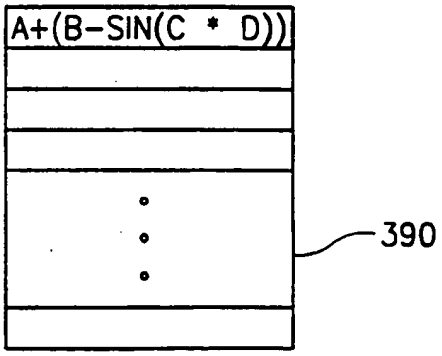



FIG.23F



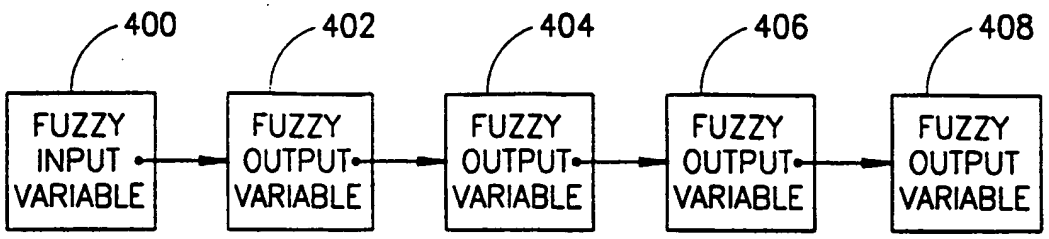

400
402
404
406
408
FUZZY INPUT VARIABLE
FUZZY OUTPUT VARIABLE
FUZZY OUTPUT VARIABLE
FUZZY OUTPUT VARIABLE
FUZZY OUTPUT VARIABLE

FIG.24

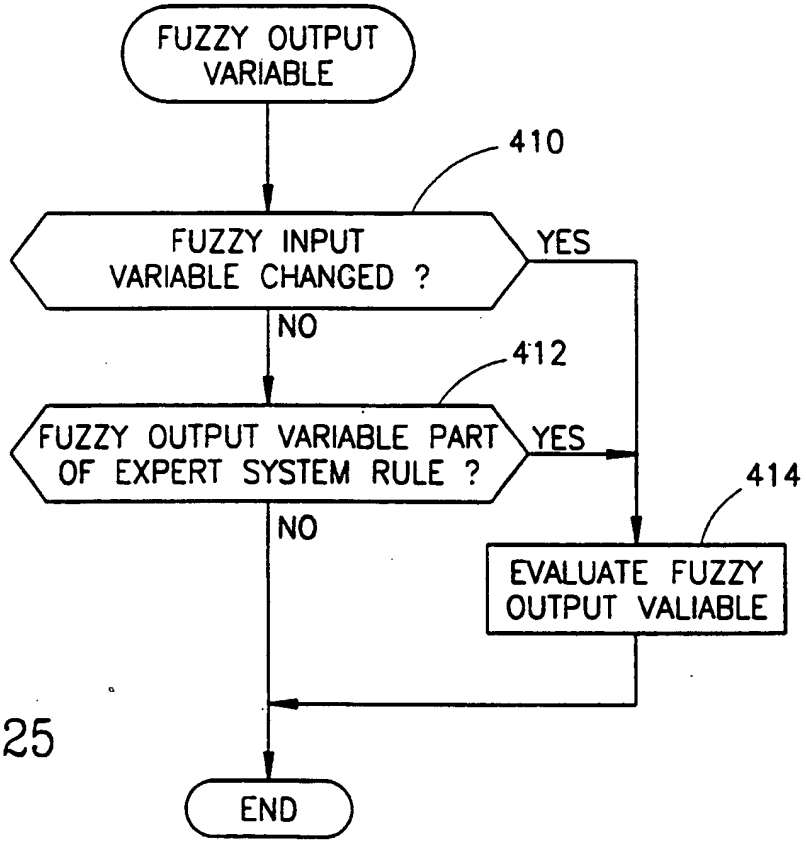

FUZZY OUTPUT VARIABLE
410
FUZZY INPUT VARIABLE CHANGED ?
YES
NO
412
FUZZY OUTPUT VARIABLE PART OF EXPERT SYSTEM RULE ?
YES
NO
414
EVALUATE FUZZY OUTPUT VALIABLE
END

FIG.25

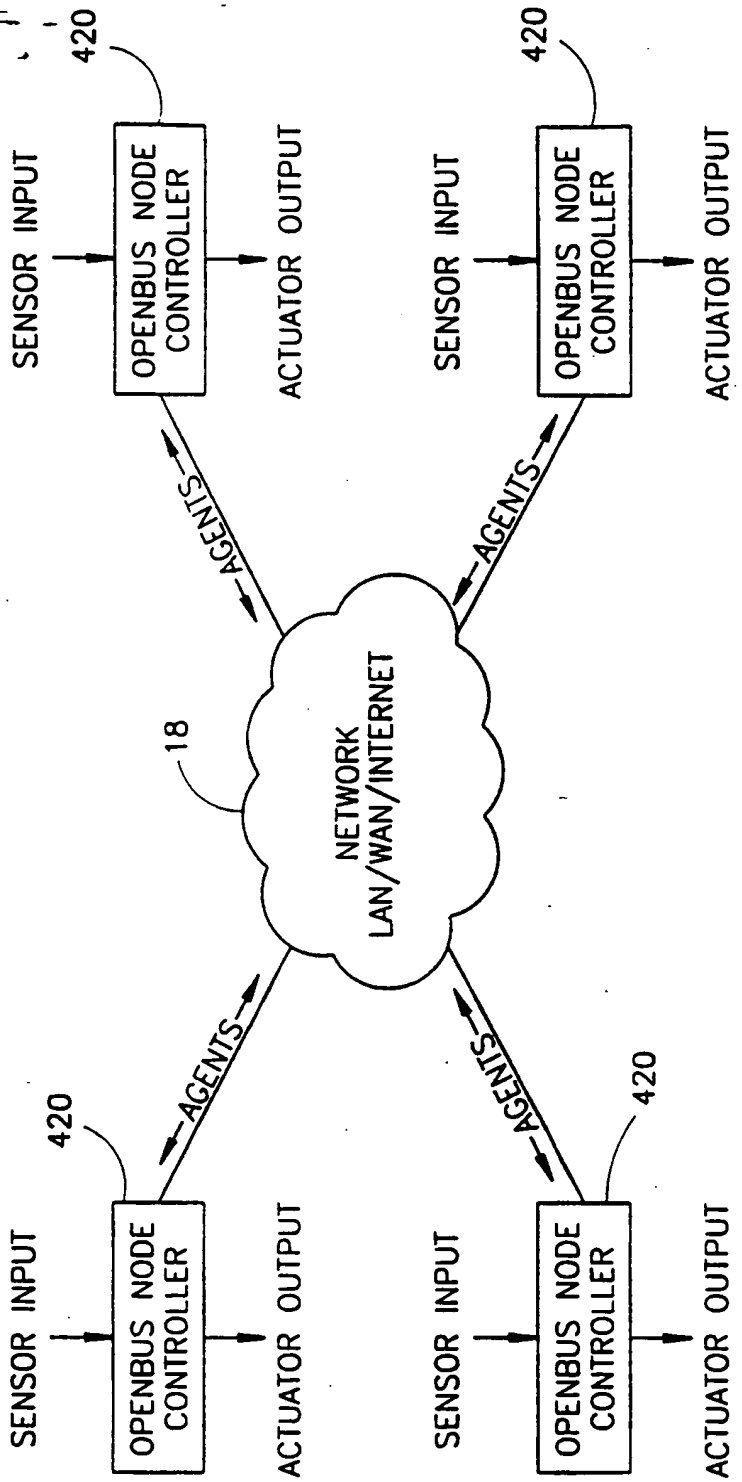

420
SENSOR INPUT
OPENBUS NODE CONTROLLER
ACTUATOR OUTPUT
AGENTS
18
NETWORK LAN/WAN/INTERNET

FIG.26

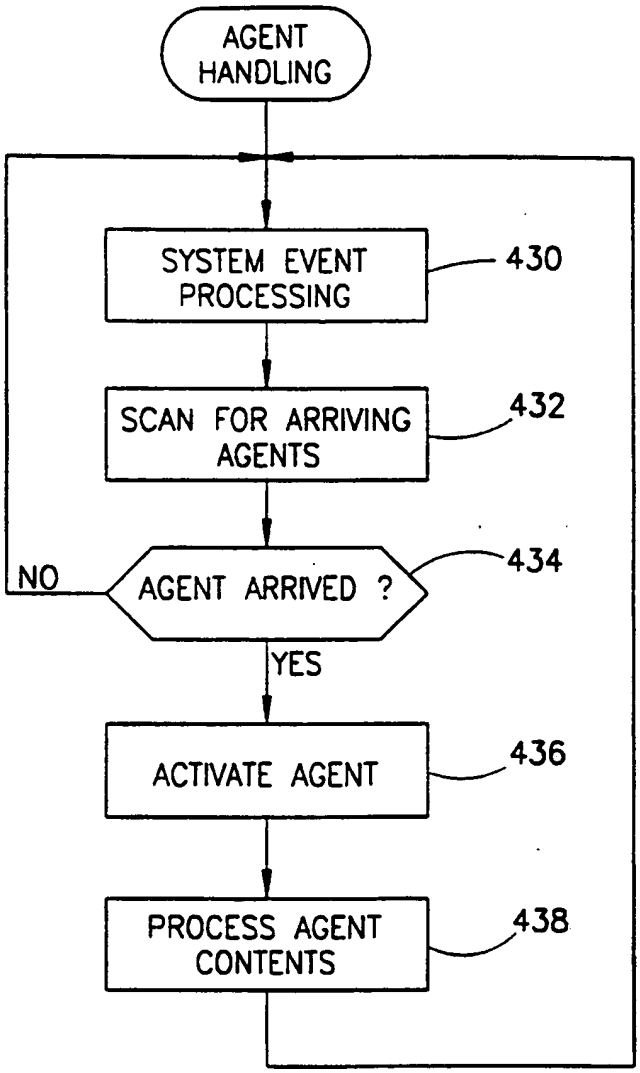

AGENT HANDLING
SYSTEM EVENT PROCESSING
430
SCAN FOR ARRIVING AGENTS
432
NO
AGENT ARRIVED ?
434
YES
ACTIVATE AGENT
436
PROCESS AGENT CONTENTS
438

FIG.27

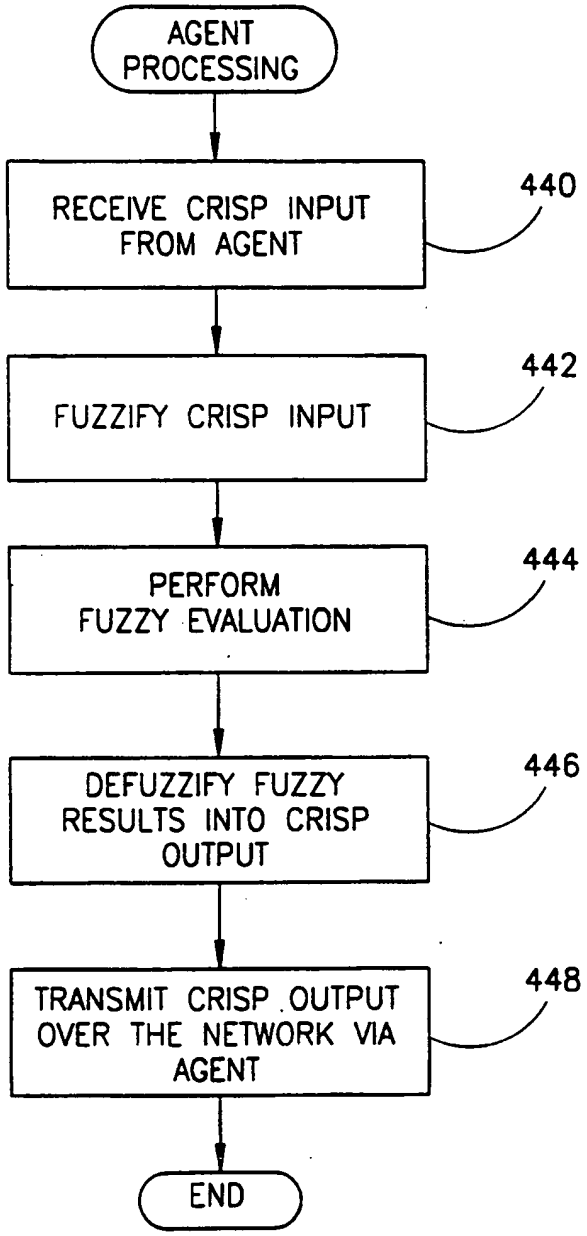

AGENT PROCESSING
RECEIVE CRISP INPUT FROM AGENT
440
FUZZIFY CRISP INPUT
442
PERFORM FUZZY EVALUATION
444
DEFUZZIFY FUZZY RESULTS INTO CRISP OUTPUT
446
TRANSMIT CRISP OUTPUT OVER THE NETWORK VIA AGENT
448
END

FIG.28

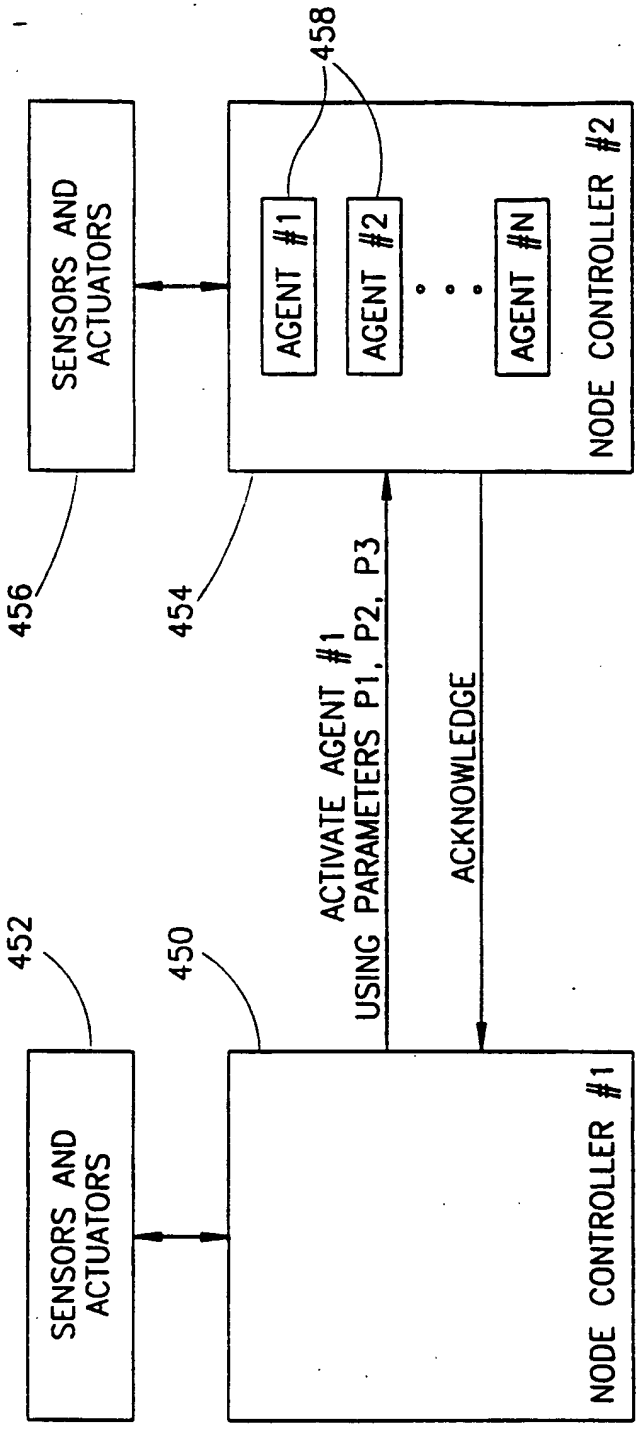

458
SENSORS AND ACTUATORS
AGENT #1
AGENT #2
AGENT #N
NODE CONTROLLER #2
456
454
ACTIVATE AGENT #1
USING PARAMETERS P1, P2, P3
ACKNOWLEDGE
452
450
SENSORS AND ACTUATORS
NODE CONTROLLER #1

FIG.29

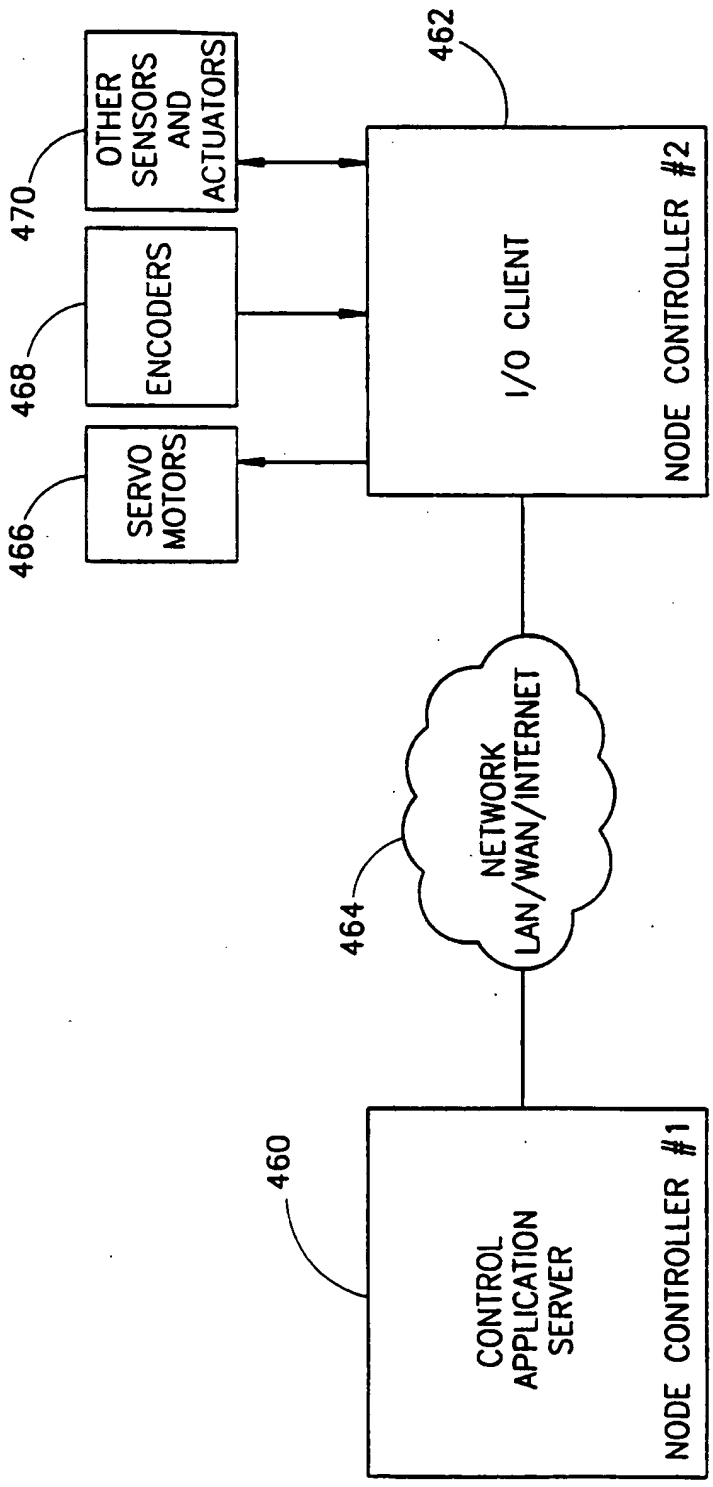

NODE CONTROLLER #1
CONTROL APPLICATION SERVER
460
NETWORK LAN/WAN/INTERNET
464
NODE CONTROLLER #2
I/O CLIENT
462
SERVO MOTORS
466
ENCODERS
468
OTHER SENSORS AND ACTUATORS
470

FIG.30

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/16 | A3

(11) International Publication Number: WO 98/36518

(43) International Publication Date: 20 August 1998 (20.08.98)

(21) International Application Number: PCT/IL98/00043

(22) International Filing Date: 29 January 1998 (29.01.98)

(30) Priority Data:
08/790,974 30 January 1997 (30.01.97) US

(71)(72) Applicants and Inventors: AZARYA, Arnon [IL/IL]; Yehuda Street 4, 47311 Ramat Hasharon (IL). AZARYA, Yitzhak [IL/IL]; Mivtza Sinai Street 5, Kiriat Ata 28000 (IL).

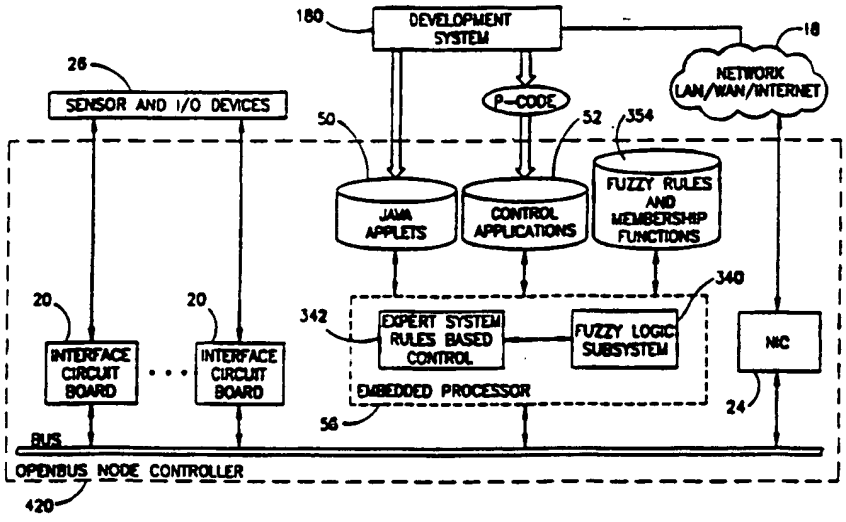
(74) Agent: EITAN, PEARL, LATZER & COHEN-ZEDEK; Lumir House, Maskit Street 22, 46733 Herzelia (IL).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

**Published**
*With international search report.*

(88) Date of publication of the international search report:
12 November 1998 (12.11.98)

(54) Title: OPENBUS SYSTEM FOR CONTROL AUTOMATION NETWORKS INCORPORATING FUZZY LOGIC CONTROL




(57)Abstract

A control system for enabling I/O boards to access communication networks for receiving and transmitting real time control information over a communication network (18) including a control bus, a node controller (10) and a development system (180). The node controller includes an expert rule based control system and a fuzzy logic subsystem. The fuzzy logic subsystem (340) is used to perform fuzzy evaluations of control data and variables. The fuzzy control system is defined by the user using fuzzy rules, variables and membership functions (354). A plurality of node controllers connected via a network can send and receive data via agents thus enabling a distributed control application. External hardware that connects to I/O devices (26) such as sensors, motors, monitors, machines, etc. can be connected to the node controller via I/O boards that receive and transmit digital signals, representing control information, to the bus. The bus allows any I/O control board (20) having a common interface, such as ISA, PCI, Compact PCI, etc., to connect to the bus by attachment to one of its slots.

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AL | Albania | ES | Spain | LS | Lesotho | SI | Slovenia |
| AM | Armenia | FI | Finland | LT | Lithuania | SK | Slovakia |
| AT | Austria | FR | France | LU | Luxembourg | SN | Senegal |
| AU | Australia | GA | Gabon | LV | Latvia | SZ | Swaziland |
| AZ | Azerbaijan | GB | United Kingdom | MC | Monaco | TD | Chad |
| BA | Bosnia and Herzegovina | GE | Georgia | MD | Republic of Moldova | TG | Togo |
| BB | Barbados | GH | Ghana | MG | Madagascar | TJ | Tajikistan |
| BE | Belgium | GN | Guinea | MK | The former Yugoslav Republic of Macedonia | TM | Turkmenistan |
| BF | Burkina Faso | GR | Greece | | | TR | Turkey |
| BG | Bulgaria | HU | Hungary | ML | Mali | TT | Trinidad and Tobago |
| BJ | Benin | IE | Ireland | MN | Mongolia | UA | Ukraine |
| BR | Brazil | IL | Israel | MR | Mauritania | UG | Uganda |
| BY | Belarus | IS | Iceland | MW | Malawi | US | United States of America |
| CA | Canada | IT | Italy | MX | Mexico | UZ | Uzbekistan |
| CF | Central African Republic | JP | Japan | NE | Niger | VN | Viet Nam |
| CG | Congo | KE | Kenya | NL | Netherlands | YU | Yugoslavia |
| CH | Switzerland | KG | Kyrgyzstan | NO | Norway | ZW | Zimbabwe |
| CI | Côte d'Ivoire | KP | Democratic People's Republic of Korea | NZ | New Zealand | | |
| CM | Cameroon | | | PL | Poland | | |
| CN | China | KR | Republic of Korea | PT | Portugal | | |
| CU | Cuba | KZ | Kazakstan | RO | Romania | | |
| CZ | Czech Republic | LC | Saint Lucia | RU | Russian Federation | | |
| DE | Germany | LI | Liechtenstein | SD | Sudan | | |
| DK | Denmark | LK | Sri Lanka | SE | Sweden | | |
| EE | Estonia | LR | Liberia | SG | Singapore | | |

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/IL98/00043

**A. CLASSIFICATION OF SUBJECT MATTER**

IPC(6) :G06F 15/16
US CL : 364/130,191; 395/200.83; 395/3; 395/701
According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/130,191,468.09,468.1; 395/3,50,60,61, 200.83,701,705,821

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A,E | US 5,757,640 A (MONSON) 26 MAY 1998, col. 9 line 40 -col. 10 line 3. | 1-24 |
| A | US 5,428,525 A (CAPPELAERE et al.) 27 JUNE 1995, col. 1 lines 7-67, col. 9 line 51-col. 10 line 3. | 1-24 |
| A | US 5,412,757 A (ENDO) 02 MAY 1995, col. 1 line 5 -col. 2 line 18. | 1-24 |
| A | US 5,043,929 A (KRAMER et al.) 27 AUGUST 1991, col. 31 lines 54-61. | 1-24 |

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

| | | | |
|---|---|---|---|
| * | Special categories of cited documents: | "T" | later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention |
| "A" | document defining the general state of the art which is not considered to be of particular relevance | "X" | document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone |
| "E" | earlier document published on or after the international filing date | "Y" | document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art |
| "L" | document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) | "&" | document member of the same patent family |
| "O" | document referring to an oral disclosure, use, exhibition or other means | | |
| "P" | document published prior to the international filing date but later than the priority date claimed | | |

| | |
|---|---|
| Date of the actual completion of the international search<br>15 JULY 1998 | Date of mailing of the international search report<br>19 AUG 1998 |
| Name and mailing address of the ISA/US<br>Commissioner of Patents and Trademarks<br>Box PCT<br>Washington, D.C. 20231<br>Facsimile No. (703) 305-3230 | Authorized officer<br>James R. Trammell<br>Telephone No. (703) 305-9600 |

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IL98/00043

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS, STN

search terms : network, control, fuzzy, expert, p-code, real-time kernel,compile